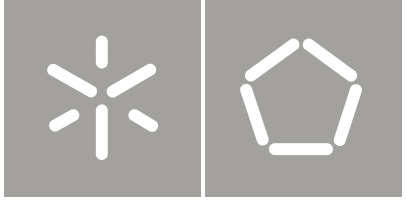
Universidade do Minho
Escola de Engenharia

Arménio António Fernandes Antunes

Análise Espacial de Grandes Quantidades de Dados de Movimento Usando Técnicas de Clustering Baseadas em Densidade

Outubro de 2012



Universidade do Minho
Escola de Engenharia

Arménio António Fernandes Antunes

Análise Espacial de Grandes Quantidades de Dados de Movimento Usando Técnicas de Clustering Baseadas em Densidade

Dissertação de Mestrado
Mestrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação de
Professora Doutora Maribel Yasmina Santos
Professor Doutor Adriano Moreira

Outubro de 2012

Agradecimentos

Este trabalho não seria possível sem o apoio de algumas pessoas, a quem quero apresentar aqui os meus mais sinceros agradecimentos.

Gostaria de agradecer à Professora Doutora Maribel Yasmina Santos e ao Professor Doutor Adriano Moreira pelo suporte e paciência que demonstraram, nunca descurando as suas tarefas de orientadores. Foi de fato um privilégio poder realizar este trabalho como seu orientando.

Gostaria de agradecer à minha família, que me apoiou incondicionalmente, como sempre foi hábito.

Finalmente, gostaria de agradecer também aos meus amigos, que nunca se lamentaram por estarem sempre comigo nos momentos de maior pressão.

A todos, o meu sincero OBRIGADO!

Resumo

A análise de entidades em movimento, representados através de *Moving Point Objects* (MPO), é útil nas mais variadas áreas, desde o estudo de migrações de animais, até ao estudo do comportamento de multidões. Grandes quantidades de dados sobre movimento continuam a ser recolhidas utilizando tecnologias como o *Global Gositioning Systems* (GPS) e informação geográfica voluntária baseado na Internet. Um grande desafio no estudo de dados sobre movimento é o tamanho cada vez mais avultado das bases de dados que são passiveis de serem analisadas.

Para analisar grandes quantidades de dados, com o objetivo de identificar padrões ou tendências nos mesmos, podem ser utilizados algoritmo de *clustering*. Estes podem ser de diferentes tipos. Dentre os mesmos, e dadas as características dos dados a analisar, foram selecionados os algoritmos baseados em densidade de pontos.

Os algoritmos de *clustering* cujos resultados se têm mostrado mais satisfatórios, como o "*sheared nearest neighbour*", tendem a não ser aplicáveis a bases de dados massivas, devido à sua complexidade ser quadrática, o que apresenta custos em termos de tempo de execução. Este trabalho propõe-se a identificar e avaliar alternativas que possam ser adotadas no sentido de diminuir a complexidade e consequentemente o tempo de execução do algoritmo.

A otimizações identificadas e implementadas baseiam-se na redução muito significativa do número de cálculos de proximidade necessários para definir as listas de vizinhos mais próximos de cada ponto. Isto foi conseguido através da divisão dos pontos, através das suas coordenadas espaciais, por uma matriz, e comparando os pontos de cada célula dessa matriz com os pontos de células vizinhas. Foram atingidos resultados relevantes quando se tornou possível reduzir o tamanho dessas células sem nenhuma restrição ao nível da ocorrência de erros de clustering.

O algoritmo resultante foi implementado numa ferramenta preparada para facilitar a análise de dados sobre movimento, e permitir o uso da estratégia desenvolvida neste trabalho noutros fins, diferentes do uso do algoritmo SNN.

Abstract

The analysis of bodies in motion, represented through Moving Point Objects (MPO), is useful in various areas, from the study of migrations of animals up to the study of behavior of crowds. Large amounts of movement data continue to be collected using technologies such as Global Gositioning Systems (GPS) and geographic information-based voluntary Internet. A major challenge in the study of movement data is the increasingly large size of databases that are ready for analysis.

To analyze large amounts of data in order to identify patterns or trends in them, a clustering algorithm can be used. These can be of different types. Among them, and given the characteristics of the data to be analyzed, density-based algorithms were selected.

The clustering algorithms whose results have proved most satisfactory, as the "sheared nearest neighbor", tend not to be applicable to massive data bases because of their quadratic complexity, which has costs in terms of runtime. This study proposes to identify and evaluate alternatives that can be adopted to reduce the complexity and thus the running time of the algorithm.

The optimizations identified and implemented are based on the significant reduction in the number of calculations needed to determine the nearest neighbors lists of each point. This was accomplished by dividing points through their spatial coordinates, into a matrix, and comparing the points of each cell of this array points to neighboring cells. Significant results were achieved when it became possible to reduce the size of these cells without any restriction in terms of the occurrence of errors in clustering.

The resulting algorithm has been implemented in a tool equipped to facilitate analysis of movement data, and enable the use of the strategy developed in this work for other purposes, different from the use of SNN algorithm.

Índice de Conteúdos

Índice de Figuras

Acrónimos

CSV	*Comma Separated Values*
FCA	*Fast Clustering Algorithm*
FDCA	*Fast Density-based Clustering Algorithm*
GPS	*Global Positioning System*
GMT	*Greenwich Mean Time*
LAC	*Local Aglomerative Characteristics*
LAD	*Local Average Distance*
LMD	*Local Maximum Distance*
MPO	*Moving Point Object*
RD	*Reachability Density*
SNN	*Sheared Nearest Neighbour*
SNNAE	*Sheared Nearest Neighbour Algorithm with Enclosures*
SGBD	Sistema de Gestão de Bases de dados

Capítulo 1
Introdução

Este capítulo contém uma introdução à problemática da falta de eficiência dos algoritmos de clustering baseados em densidade de pontos, nomeadamente, o *Sheared Nearest Neighbour* (SNN). Em primeiro lugar procura-se enquadrar o trabalho efetuado e esclarecer a motivação por detrás do mesmo. De seguida, são apresentados os objetivos e resultados esperados. Passa-se depois à definição da abordagem metodológica, terminando numa breve descrição da estrutura do documento.

1.1 Enquadramento e Motivação

A crescente disponibilidade de redes móveis e dispositivos móveis tem facilitado a recolha de dados sobre o movimento, independentemente de estarmos a falar do movimento de um pedestre, de um indivíduo num automóvel, de um ciclista, de um barco, de um avião, entre outros. O estudo destes dados pode ser muito vantajoso, possibilitando a descoberta de padrões úteis para as mais variadas áreas, seja gestão de veículos ou frotas, estudo do comportamento de multidões ou estudo de hábitos de animais (Ertoz, Steinbach, & Kumar, 2002).

A facilidade com que é efetuada a recolha destes dados de movimento, o número

de indivíduos observados, e a abrangência geográfica alargada, conduzem facilmente a conjuntos enormes de registos de movimento (frequentemente na ordem dos milhões de registos). A forma como os dados são representados ou agregados permite realizar um conjunto de análises e explorar a forma como o movimento ocorre. Em particular interessa verificar como o movimento é influenciado por fatores externos como a hora do dia, o local em análise, etc. Poderá ser importante identificar como o contexto em que o movimento ocorre influencia os padrões de movimento, e conhecer os percursos mais usuais e o seu padrão de utilização. Neste contexto avaliar a qualidade dos dados disponíveis para análise, nomeadamente a precisão e acuidade com que são recolhidos, é muito importante.

Para estudar dados espaciais são habitualmente usadas técnicas de d*ata mining* como o agrupamento baseado na densidade de pontos. Exemplos mais comuns deste tipo de ferramentas são o algoritmo *Shared Nearest Neighbour* (SNN) (Ertöz, Steinbach, & Kumar, 2003) ou o algoritmo DBSCAN (Ester, Kriegel, Sander, & Xu, 1996).

As técnicas de *clustering,* como as referidas anteriormente, implicam o cálculo da similaridade de cada objeto presente no conjunto de dados calculando a sua distância em relação a todos os outros objetos. Isto significa que, para as técnicas mais comuns de *clustering* por densidade de pontos, o tempo de processamento vai aumentar quadráticamente com o aumento da quantidade de dados, surgindo também dificuldades de gestão da memória necessária para armazenar os dados. Torna-se fundamental encontrar mecanismos que possam otimizar o uso da capacidade computacional e capacidade de memória disponível nos computadores utilizados para efetuar as tarefas de análise.

Para além da posição espacial dos objetos, é frequente que cada um apresente outras dimensões para análise (como a direção ou a velocidade do movimento), as quais podem ser consideradas na identificação de padrões de movimento. No processo de extração de conhecimento a partir deste tipo de dados, poderá ser importante analisar estas outras dimensões dos mesmos, aumentando assim a complexidade do processo e deteriorando o tempo de processamento.

1.2 Objetivos da Dissertação e Resultados Esperados

Enquadrada a área de trabalho e sua motivação, importa referir a questão de investigação que guia o trabalho a desenvolver. Pretende-se, assim, dar resposta à questão: Como lidar com grandes quantidades de dados nas técnicas de *clustering* baseadas na densidade de pontos em dados georreferenciados?

Sendo assim, este trabalho inclui o estudo do algoritmo de análise de dados baseado na densidade de pontos, SNN, no sentido da identificação de estratégias para lidar com grandes volumes de dados. Estas estratégias deverão ser implementadas e testadas, de forma a alterar o algoritmo, reduzindo o tempo de processamento de dados georreferenciados.

Posteriormente, o algoritmo resultante deverá ser implementado numa aplicação, de forma a poder ser utilizado na tarefa de clustering de dados sobre movimento.

1.3 Abordagem Metodológica

Trata-se de um trabalho cujo intuito passa por desenvolver formas de resolver ou minimizar problemas do mundo real, esperando produzir algum contributo para a base teórica da área de *data mining* de dados georreferenciados. Este contributo pressupõe a construção de algoritmos, métodos e software, que serão testados e avaliados, produzindo resultados quantitativos.

Mais especificamente, será realizada uma revisão de literatura na área de análise de dados sobre movimento. Concretamente, pretende-se estudar e compreender o problema da análise de grandes volumes de dados georreferenciados, especificamente dados sobre movimento, com técnicas de *clustering*, dando ênfase aos algoritmos baseados em densidade de pontos.

Numa fase posterior, serão concetualizadas uma série de estratégias com o intuito de dar resposta ao problema da ineficiência do processo de análise de dados

sobre movimento, no que diz respeito ao tempo de processamento.

Seguir-se-á a implementação e teste de melhorias ao nível do desempenho do algoritmo. Para cada desenvolvimento, deverão ser quantificados e avaliados resultados, quer no pré-processamento dos dados, quer na própria otimização de algoritmos existentes. Esta avaliação deverá identificar claramente os respetivos pontos onde, e de que forma,o desempenho é afetado, face a critérios e contextos pré-estabelecidos

As implementações necessárias serão desenvolvidas em java, devido à sua independência quanto ao sistema operativo utilizado, e ao facto da sua orientação a objetos facilitar o desenvolvimento modular e reutilização de código. Para visualização de resultados serão usadas folhas de cálculo e as ferramentas ArcGIS.

1.4 Estrutura do Documento

O presente documento está organizado em 6 capítulos. O primeiro é a introdução, onde também a secção presente se insere, contextualizando-se o trabalho e abordando motivação, pertinência, objetivos e resultados esperados.

No segundo capítulo, o enquadramento conceptual, são explorados todos os temas considerados relevantes até à data para a dissertação, revendo-se a literatura considerada mais representativa da área, procurando explanar o contexto cientifico onde este trabalho se enquadra.

O quarto capítulo, “Clustering de Dados Geo-referenciados”, aborda, em primeiro lugar, uma implementação e teste do algoritmo SNN, que serviu para estudar as suas características de eficiência. De seguida são propostas adições a este algoritmo e são apresentados resultados de desempenho para essas alterações, de forma a que seja possível avaliar cada solução desenvolvida.

O quinto capítulo retrata uma aplicação desenvolvida para a utilização do algoritmo resultante dos passos descritos no capítulo anterior. Não se centrando a descrição apenas na sua arquitetura, é focada também à sua interface e funcionalidade, bem como o seu modo de utilização.

Capítulo 2
Enquadramento Conceptual

Em toda a sua existência, as pessoas observaram e demonstraram interesse em entidades que se movem, desde insetos ou peixes a planetas e estrelas, e investigaram os seus padrões de movimento. Estes podem ser estudados sob o ponto de vista de vários aspetos, como trajetórias no espaço, velocidade, direção, e as suas dinâmicas ao longo do tempo (N. Andrienko, Andrienko, Pelekis, & Spaccapietra, 2008).

Neste capítulo serão focados conceitos relacionados com dados sobre movimento, e análise dos mesmos. Será abordado o conceito de clustering e é efetuada uma revisão aos tipos mais comuns, com especial ênfase nos algoritmo de densidade de pontos. Finalmente, interessa também perceber a evolução, e o que tem sido prática comum na área de análise de dados sobre movimento.

2.1 Dados Sobre Movimento

Segundo Andrienko et al. (2008), as caraterísticas do movimento, na análise do mesmo, podem ser consideradas face à trajetória, espaço, tempo, e às próprias atividades da entidade e a eventos relacionados. Dados sobre movimento podem conter qualquer uma, ou várias destas caraterísticas abstraídas em si.

Trajetória é o caminho efetuado por uma entidade no espaço por onde se move. Cada ponto neste caminho representa uma posição no espaço e um instante no tempo

(Alvares, Bogorny, de Macedo, Moelans, & Spaccapietra, 2007). Desta forma, tempo é uma aspeto indissociável de uma trajetória.

Também é necessário um sistema referencial para representar o tempo nos dados. É comum, para referenciação, utilizar-se o calendário Gregoriano, tal como a divisão do dia em horas, horas em minutos, e assim sucessivamente. O período do dia pode ser especificado de acordo com o local onde os dados são recolhidos ou de acordo com o *Greenwich Mean Time* (GMT). Os dados podem ser referenciados de forma relativa, começando a contar o tempo, por exemplo, a partir do momento em que se efetuou o primeiro registo.

Como o tempo físico é uma dimensão contínua, a única forma de se tentar perceber o que sucede entre dois instantes presentes nos dados, é realizar uma estimativa através de interpolação.

A heterogeneidade das propriedades do tempo, normalmente, não é explicitamente refletida nos dados, portanto, não pode ser automaticamente levada em consideração na análise dos mesmos. A utilização dos dados temporais na análise de dados depende muito do conhecimento prévio do analista (N. Andrienko et al., 2008).

2.2 Processo de Análise de Dados

Segundo Miller & Han (2009), o processo de descoberta de conhecimento em bases de dados envolve, tipicamente, um conjunto de passos fundamentais, agrupados em categorias de atividades mais genéricas:

- Pré-processamento dos dados
- Data Mining
- Construção de Conhecimento

O conhecimento prévio refere-se à compreensão do domínio da aplicação. O pré-processamento de dados implica a seleção de dados onde se vai concentrar a exploração de padrões, a limpeza dos dados, a remoção de dados que representam ruído,

e a redução de dados, englobando transformações, projeções e agregações, potenciando a identificação de representações úteis dos dados na fase de construção de conhecimento.

A atividade seguinte, *data mining,* implica selecionar o tipo de tarefa a realizar, isto é, escolher que tipo de padrão se procura nos dados, como classes, associações, regras, ou *clusters*. Depois, interessa escolher a melhor técnica de *data mining*, ou seja, a mais apropriada ao tipo de padrão que se pretende explorar. Finalmente, é aplicada a técnica apropriada para procurar os padrões de interesse.

A última atividade, construção de conhecimento, engloba a interpretação dos padrões descobertos, a construção de um modelo representativo da realidade, muitas vezes através de visualização, e a consolidação do conhecimento obtido, seja através da incorporação do mesmo em sistemas informáticos, ou através da documentação e entrega de um relatório às entidades interessadas.

Os processos de análise de dados, recorrendo a algoritmos de *data mining*, podem ser divididos entre exploratórios ou confirmatórios, dependendo da existência de modelos apropriados para os dados em estudo. Mas um elemento essencial em ambos os casos é o agrupamento ou classificação, seja baseado no agrupamento natural (*clustering*) revelado através de análise exploratória, ou no grau de encaixe (*fitting*) num modelo existente (Jain, Murty, & Flynn, 1999).

A representação dos dados sobre movimento é um aspecto essencial para que seja possível perceber os fenómenos inerentes aos mesmos (Gennady Andrienko, Andrienko, & Wrobel, 2007). Os métodos mais comuns de representação de movimento são setas ou fluxos desenhados num mapa, técnica de cubo espaço-temporal, mapas animados e cubos interactivos. No entanto métodos puramente visuais falham quando é necessário examinar movimento de um grande número de entidades ou espaços temporais muito amplos. A maioria das abordagens para lidar com grandes volumes de dados envolvem processos de agregação, como o *clustering.*

A análise por *clustering* é a organização de um conjunto de elementos de dados, dividindo-os por conjuntos, baseados na similaridade entre os mesmos. Elementos de

um determinado conjunto (*cluster*) resultante da análise, serão mais similares entre si do que em relação a um elemento pertencente a outro *cluster* (Jain et al., 1999). Esta análise tem o propósito de facilitar a compreensão dos dados (Ertöz et al., 2003).

No processo de *clustering* persistem diversos desafios relacionados com qualidade dos resultados encontrados, tais como identificar *clusters* com tamanhos, densidades e formas diversas, como lidar com ruído e *outliers*, e como determinar o número de *clusters* (Ertöz et al., 2003).

São ainda reconhecidos desafios relevantes no *clustering* devido à existência recente de grandes quantidades de dados com elevado número de dimensões, bem como à capacidade computacional para os processar (Mennis & Guo, 2009).

A atividade de *clustering* para a identificação de padrões envolve alguns passos comuns a todos os tipos de técnicas existentes (que são detalhados na próxima secção). Em primeiro lugar considera-se a representação que se pretende obter, como por exemplo, o número de agrupamentos que irão existir (caso o algoritmo exija que se defina esta variável no início), quais as dimensões fornecidas ao algoritmo para processamento e quais as suas escalas ou tipo. De seguida é definida a forma de medição de proximidade entre objetos de dados mais apropriada para o âmbito dos dados em estudo. Esta medição é geralmente efetuada através de uma função de distância entre pares de objetos de dados (Jain et al., 1999). Uma função de distância geralmente utilizada nos dados sobre movimento é a distância euclidiana, que se trata de um cálculo de distância utilizando apenas as coordenadas espaciais dos pontos (Santos, Silva, Moura-Pires, & Wachowicz, 2012).

Dado este trabalho se focar na melhoria de desempenho do algoritmo de *clustering* SNN, interessa explorar e conhecer os algoritmos já existentes e as suas estratégias, visto poderem-se revelar úteis na alteração ao SNN. É também importante explorar outras abordagens que já se tenham baseado no SNN, com a preocupação de aumentar a eficiência do algoritmo.

Na secção seguinte será então abordado o universo dos tipos de algoritmos de clustering mais comuns, aprofundando-se um pouco mais a parte respeitante aos algoritmos de densidade de pontos.

2.3 Algoritmos de *Clustering*

Existe uma variedade considerável de algoritmos de *clustering*. Estes algoritmos podem ser classificados em diferentes tipos, com base na estratégia adotada para realizar o agrupamento.

De seguida serão abordados os tipos de algoritmos de *clustering* mais comuns na literatura.

2.3.1 Algoritmos Hierárquicos

Os métodos hierárquicos agrupam os dados numa estrutura que pode ser abstraída numa árvore de *clusters*. Existem dois tipos principais de métodos desta natureza. Os métodos aglomerativos, que formam os *clusters* de baixo para cima até que todos os elementos dos dados pertençam ao mesmo *cluster*, ou seja, começa por encontrar as folhas só depois chegando à raiz (as folhas correspondem a registos individuais que vão sendo sucessivamente agrupados). Já os métodos divisivos processam-se de forma contrária, agrupando o conjunto total de dados como um único *cluster*, e subdividindo o mesmo em subníveis, começando na raiz e acabando nas folhas (Kotsiantis & Pintelas, 2004).

Os algoritmos deste tipo apresentam como vantagens o facto de serem rápidos e não necessitarem de conhecer previamente o número de *clusters* que se irão obter (Kotsiantis & Pintelas, 2004). Estes usam vários critérios para decidir, a cada passo, quais os *clusters* que devem ser unidos, no caso dos de tipo aglomerativo, ou divididos, no caso de serem de tipo divisivo. Assim sendo, a utilização deste tipo de algoritmos implica a definição prévia desses mesmos critérios.

Para técnicas hierárquicas aglomerativas, o critério é, normalmente, unir os *clusters* que estejam mais próximos entre si. Essa proximidade entre dois *clusters* pode ser definida pela similaridade entre os elementos mais similares (*single-link*), ou menos similares (*complete-link*), entre si, cada um pertencente a um dos *clusters*. Existe também a possibilidade de se usar uma mistura das duas definições de proximidade anteriores (*average-link*) (Kotsiantis & Pintelas, 2004).

O método de *clustering* aglomerativo de ligação única (*single-link*) é o algoritmo mais apropriado para capturar *clusters* com formas não esféricas. No entanto é muito suscetível a ruído e não consegue lidar com *clusters* de densidades diversas. Os outros algoritmos de aglomeração, de ligação completa (*complete-link*) ou de média de grupo (*average-link*), não são tão afetados pelo ruído, mas têm tendência a encontrar *clusters* esféricos (Ertöz et al., 2003).

Dos algoritmos hierárquicos mais comuns, são exemplo o Cure (Guha, Rastogi, & Shim, 1998) ou o Chameleon (Karypis, Han, & Kumar, 1999).

O algoritmo Cure usa um conjunto predefinido de elementos de dados representativos, em vez de usar um centroide. A similaridade entre dois *clusters* é medida pela similaridade entre o par de pontos representativos mais próximos, cada um pertencente a um *cluster* diferente. Ao contrário dos algoritmos que usam centroides ou ponto médio, este permite encontrar *clusters* com diferentes formas, sejam elipsóides, ou expirais, entre outras (Guha et al., 1998).

O algoritmo Chameleon encontra os *clusters* nos dados através de duas fases. Primeiro calcula os K vizinhos mais próximos de cada ponto, produzindo um grande número de pequenos *clusters*. Depois, usa um algoritmo aglomerativo para combinar estes sub-*clusters* repetidamente (Karypis et al., 1999).

2.3.2 Algoritmos de Partição

Existem dois conjuntos principais de algoritmos de partição, baseados nos centroides e baseados nos pontos médios. No primeiro conjunto, cada *cluster* é representado pelo

centro de gravidade dos seus elementos. No segundo conjunto, os algoritmos que utilizam os pontos médios representam cada *cluster* através dos elementos mais próximos ao centro de gravidade (Kotsiantis & Pintelas, 2004).

Um dos mais significativos métodos de partição centroide é o k-means (Kotsiantis & Pintelas, 2004). Este é fácil de implementar e a sua complexidade é linear, ou seja, O(n).

O algoritmo k-means necessita da especificação inicial do número de *clusters* que serão encontrados. Em primeiro lugar, encontra aleatoriamente os elementos que começarão por ser centros de *cluster*. Depois, cada elemento de dados será atribuído ao *cluster* cujo elemento central lhe for mais próximo.

É um algoritmo de *clustering* muito usado. Não é muito eficaz quando utilizado em dados que contenham elementos muito aleatórios, e que não deviam ser agrupados nos *clusters* porque lhes vão retirar significado, ou quando utilizado em dados cujos *clusters* teriam naturalmente tamanhos ou formas diversas (Ertöz et al., 2003).

2.3.3 *Clustering* Baseado em Grelha

A ideia base destas técnicas é herdar a topologia do espaço de atributos subjacente (Berkhin, 2006). Ou seja, o espaço de *clustering* é dividido em células ou segmentos, onde são efetuadas operações específicas, como o produto cartesiano de determinados atributos dos elementos de dados. Essas células são depois ligadas para formar *clusters,* de acordo com a sua densidade (Kotsiantis & Pintelas, 2004). Este tipo de *clustering* utiliza, portanto, uma forma alternativa ao *clustering* baseado em densidade, para lidar com conceitos de densidade, conetividade ou fronteira entre elementos de dados.

2.3.4 Algoritmos Baseados em Densidade de Pontos (*Density-Based*)

Este tipo de algoritmos pretendem encontrar *clusters* baseados na densidade de pontos de dados numa determinada região. Ou seja, na sua forma mais simples, para um determinado elemento de um *cluster*, a sua vizinhança limitada por um determinado raio (Eps), tem de ter um número de elementos mínimo (MinPts) (Kotsiantis & Pintelas, 2004). Segundo Kotsiantis & Pintelas (2004), um dos algoritmos mais conhecidos deste tipo é o DBSCAN.

Algoritmo DBSCAN

O algoritmo DBSCAN (Ester et al., 1996) foi especialmente pensado para bases de dados espaciais. Segundo Tripathy, Maji, & Patra (2011), este algoritmo pode ser considerado a base de todos os algoritmos baseados em densidade de pontos.

Ester et al. (1996) definem um *cluster* à custa da sua composição. Na mesma, estes integram um conjunto de pontos nucleares (*core points*) rodeados por pontos de fronteira (*border points*). Os pontos que não satisfaçam determinados requisitos para entrarem num *cluster* são considerados ruído (*noise points*).

Para cada ponto é contabilizado o número de pontos que se encontram a menos de uma determinada distância (Eps) de si. Se dessa contabilização resultar um número maior que um valor (MinPts) pré-estabelecido, o ponto é então considerado *core.*

Os pontos de fronteira (*border*) são todos aqueles que, não possuindo uma quantidade mínima de pontos (MinPts) com uma distância inferior a Eps de si, situam-se eles próprios a uma distância inferior a Eps de um ponto *core*. Na 2.1, os pontos “P” e “Q” são ambos exemplos de pontos de fronteira.

Todos os pontos que, não sendo *core*, não possuam nenhum ponto nuclear a uma distância inferior Eps de si, são considerados pontos de ruído (*noise*).

Os *clusters* são definidos a partir do encadeamento de relações entre os pontos nucleares e os pontos a uma distância inferior a Eps de si. Ou seja, como é possível visualizar na 2.1, um ponto A é atribuído a um determinado *cluster* se na sua vizinhança,

a uma distância inferior a Eps, estiver um ponto B nuclear, já pertencente a esse mesmo *cluster*. Posteriormente, se o ponto A for ele também nuclear, o mesmo acontece aos pontos cuja proximidade a A é menor que Eps, sendo este ponto acessível por densidade a partir do ponto B. Os pontos de proximidade menor que Eps de A, ou seja, conectados a A por densidade, não sendo nucleares, passam então a ser pontos fronteira do *cluster*.

O algoritmo começa a construir um *cluster* a partir de um ponto nuclear aleatório, como por exemplo, a partir do ponto "W" na Figura 2.1, agrupando no mesmo todos os pontos com os quais este se relacione. Quando mais nenhum ponto tem relação de densidade com o primeiro, volta-se a escolher aleatoriamente um ponto nuclear, que não tenha ainda sido atribuído a nenhum *cluster*, e repete-se novamente o processo até que não existam mais pontos nucleares sem *cluster*.

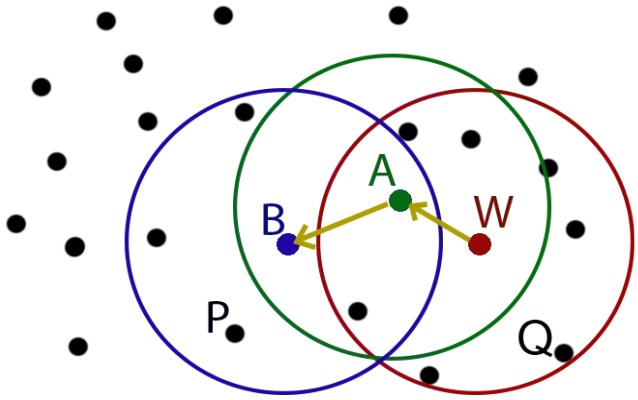


Figura 2.1: Conetividade por densidade.

O algoritmo DBSCAN lida muito bem com *clusters* de tamanhos e formas variadas, no entanto, não é eficaz com *clusters* de diferentes densidades (Ertöz et al., 2003; Moreira, Santos, & Carneiro, 2005).

Algoritmo *Sheared Nearest Neighbour* (SNN)

O algoritmo SNN (Ertöz et al., 2003) consegue encontrar *clusters* de variados tamanhos, formas ou densidades, mesmo com a presença de grandes quantidades de

pontos de ruído. Tem também a capacidade de lidar com dados que contenham um grande número de dimensões, e determina automaticamente o número de *clusters*.

Este algoritmo diverge do DBSCAN no cálculo da densidade. Mantém-se a noção de ponto nuclear *(core),* ponto de fronteira *(border)* ou ponto de ruído (*noise*). No entanto, os parâmetros que definem se um determinado ponto é considerado nuclear são diferentes.

Primeiro, são calculados os K pontos mais próximos de cada ponto, esses serão referidos como seus vizinhos. Depois, para cada ponto, encontra-se o número de vizinhos que partilhem mais de Eps vizinhos consigo próprio, contabilizando assim a densidade. Finalmente, se a densidade de um ponto for maior que MinPts, esse ponto é considerado nuclear. O processo de construção dos clusters é, a partir daqui, semelhante ao DBSCAN.

Sheared Nearest Neighbour Algorithm with Enclosures

O algoritmo SNN, embora satisfaça todos os requisitos comuns do *clustering* de pontos, não tem bom desempenho quando se tratam de grandes bases de dados. A complexidade deste algoritmo é de $O(n^2)$, isto devido à necessidade de calcular a matriz de similaridade entre todos os pontos (Bhavsar & Jivani, 2009).

Para otimizar o algoritmo e diminuir o tempo de processamento necessário para um determinado conjunto de pontos, Bhavsar & Jivani (2009) conceptualizaram o "*Sheared Nearest Neighbour Algorithm with Enclosures*" (SNNAE). Este algoritmo usa uma abordagem por delimitação, dividindo o conjunto de dados em conjuntos de dados mais pequenos (os quais apresentam alguma sobreposição). Deste modo, reduz-se o tempo de processamento do algoritmo sobre os dados, produzindo matrizes de distância apenas sobre os pontos do mesmo espaço delimitado.

A grande limitação deste algoritmo prende-se com o facto de, ao delimitar-se o espaço, pode afetar-se a similaridade relacionada com outras dimensões. Ao serem incluídas outras dimensões na medição de similaridade, como por exemplo a direção, pode-se gerar erros nos resultados do *clustering*.

Fast Clustering Algorithm (FCA)

Este algoritmo propõe-se a encontrar clusters com diferentes tamanhos, formas e densidades, característica herdada do SNN, pretendendo apresentar uma complexidade quase linear (Li, Jiang, & Su, 2009).

O FCA utiliza o conceito de "vizinho mais próximo", mas não de uma forma direta como o algoritmo SNN. Em primeiro lugar é aplicado um algoritmo de passagem única, particionando o *dataset* em K *clusters*. Todos estes *clusters* têm aproximadamente o mesmo raio.

A segunda fase deste algoritmo trata cada *cluster* previamente obtido como um objeto, e une-os agora com uma versão melhorada do algoritmo SNN. Esta versão tem o propósito de ter maior eficácia com dados em espaços discretos, ou seja, não contínuos, relativamente ao SNN original.

No entanto, este algoritmo precisa de mais parâmetros de entrada. Além de K, Eps e MinPts, também fornecidos originalmente ao SNN, é necessário definir o raio "r", um parâmetro necessário ao algoritmo de passagem única, usado para restringir o raio de cada cluster. Este fator aumenta a dependência da qualidade dos *clusters* finais na experiência anterior do analista, bem como nas características dos próprios dados.

Quando, na segunda fase, é aplicado o SNN, os *clusters* iniciais apenas podem ser unidos, quando, na realidade, poderia ser necessário a divisão de qualquer um desses *clusters* entre outros *clusters*. Será então duvidoso que os resultados deste algoritmo sejam sempre os mais corretos. Além deste fator, este algoritmo parece não ser ideal para lidar com mais do que a dimensão espacial, visto existir a definição de uma raio para a primeira parte do algoritmo.

No que toca à eficiência deste algoritmo será benéfico, de fato, reduzir a quantidade de elementos com que o SNN terá que lidar. No entanto falta perceber se o ganho será realmente substancial, dado o custo inicial de formação dos clusters de pequena dimensão.

Local Agglomerative Characteristics (LAC)

O algoritmo baseado em *Local Agglomerative Characteristics* (Niu, Yang, & Fu, 2010) pretende colmatar algumas limitações presentes no SNN. Nomeadamente o uso de um critério de classificação de objetos muito restritivo, sendo confinado apenas aos K vizinhos mais próximos, ou o facto da união ou separação entre dois *clusters* poder depender de uma única ligação.

Este algoritmo define dois parâmetros, *Local Maximum Distance* (LMD) e *Local Average Distance* (LAD). Por sua vez, estes deverão refletir as caraterísticas da distribuição de dados num determinado local, no entanto os autores optam por usar apenas o LAD, visto que o LMD é demasiado sensível à forma dos *clusters.*

Na primeira fase, o algoritmo calcula o grafo de similaridade SNN, com limites impostos pela LAD, originando a matriz de similaridade. Por cada par de objetos de dados, encontra no *dataset* os seus K vizinhos. Estes podem ser vistos como um *cluster* local com caraterísticas aglomerativas elevadas. Depois, calcula o LAD para controlar a geração do grafo de similaridade.

Numa segunda fase, encontra os elementos conectados através das relações de densidade, aplicando um parâmetro limite de similaridade, e, ao mesmo tempo, ajusta dinamicamente a associação ao *cluster.*

Os autores sugerem que este algoritmo poderá ser mais eficiente que o SNN, no entanto não o demonstram, parecendo haver ainda necessidade de algum desenvolvimento.

Fast Density-based Clustering Algorithm

O *Fast Density-based Clustering Algorithm* (FDCA) (Tripathy et al., 2011) é um algoritmo especialmente pensado para bases de dados espaciais, tal como o algoritmo no qual se baseia, o DBSCAN.

O algoritmo DBSCAN necessita inicialmente dos parâmetros Eps e MinPts. Por sua vez, este algoritmo requer apenas o valor MinPts, que deverá ser fornecido antes da

análise. O parâmetro Eps é ajustado ao longo do processo, desta forma, além de se reduzir o envolvimento do analista, é possível identificar *clusters* locais que estejam muito próximos.

Em primeiro lugar, é definido o novo conceito de *Reachability Density* (RD). Esta representa a densidade envolvente de um ponto. Uma alta densidade de acessibilidade significa uma alta densidade em torno de um ponto.

É criado um novo cluster para um ponto se possui alta RD, e a sua vizinhança de Eps não intersecta os pontos de outro *cluster* previamente *criado*. Existem também as noções de ponto acessível ou conectado a outro ponto, análogas ao DBSCAN.

Para um determinado valor MinPts, os pontos no interior de um *cluster* têm uma RD mais alta que os pontos que se situam perto das fronteiras do *cluster*.

Neste algoritmo, a construção do *cluster* começa do interior para o exterior. Ao longo da expansão de um *cluster*, o valor de RD dos seus pontos vai diminuindo, consequentemente o valor de Eps dos mesmos pontos também reduz, evitando assim incluir pontos pertencentes a outro *cluster* próximo.

Para melhorar o desempenho deste algoritmo é aplicada uma heurística, ou seja, aplica-se a regra de que não há necessidade de processar todos os pontos presentes no círculo de raio Eps centrado num ponto de um *cluster*, mas sim apenas os que se encontram na fronteira desse círculo. Deste modo, uma menor quantidade de pontos são processados, aumentando significativamente a eficiência deste algoritmo.

2.4 Análise de Dados sobre Movimento

Meratnia e By (2002) propuseram diferentes abordagens de clustering para lidar com séries temporais multi-dimensionais e ruído. Numa primeiro abordagem a informação discreta das trajetórias é processada, de forma a ser transformada em posições temporais de uma trajetória de objetos em movimento. O período de amostragem das posições das trajetórias é constante e síncrono entre todas elas. Depois estas posições são agrupados através de um limite de similaridade. Ou seja, se duas posições são mais similares que

um limite pré-estabelecido, então são agrupadas.

A abordagem descrita não é satisfatória devido à não transitividade entre trajetórias. Isto é, se uma determinada trajetória for similar a uma segunda trajetória, e esta, por sua vez, for similar a uma terceira, não implica que a primeira e a terceira sejam similares, sendo por isso difícil gerar classes similares de trajetórias. Além deste problema, o tempo de processamento das similaridades entre as posições teria complexidade $O(n^2)$.

Numa segunda abordagem, para ultrapassar esta dificuldade, Meratnia e By (2002) propuseram a divisão das dimensões espaciais numa matriz. O limite de similaridade entre posições é agora definido pelo tamanho dado às quadrículas. A similaridade entre trajetórias é definida pelo número de vezes que cada uma visita cada quadricula. As vantagens desta abordagem seriam a independência das trajetórias individuais, fácil generalização da informação, e menor número de cálculos a efectuar, tendo por isso melhor desempenho. Ainda noutra abordagem, o mesmo principio poderá ser aplicado em clustering espacial-temporal, mas agora a similaridade seria calculada através do número de visitas de uma trajetória a uma determinada quadrícula, num determinado intervalo de tempo.

Para o caso de *clustering* de trajetórias através dos vizinhos mais próximos (SNN), Vlachos et al. (2002) identificaram vários desafios que a função de distância deveria superar. Nomeadamente, no que concerne a como lidar com diferentes frequências de amostragem, deteção de movimentos semelhantes em diferentes regiões do espaço, como lidar com ruído, e como permitir um cálculo de similaridades eficiente. Assim, para atingir estes objetivos, propuseram uma função de similaridade baseada na maior subsequência comum (Longest Common Subsequence – LCSS), demonstrando que lida melhor com dados com ruído. É usado um algoritmo hierárquico para indexar as trajetórias, dando assim resposta a *queries* relacionadas com vizinhos mais próximos, mais eficientemente. Chen et al. (2005) definem uma nova função de distância, o *Edit Distance on Real sequence* (EDR), em que a similaridade entre duas trajetórias se baseia na quantidade de adições, remoções e alterações que é necessário fazer à primeira para que fique igual à segunda. Provam também que esta função ultrapassa outras funções de

distância, como LCSS, no que toca à robustez relativamente ao ruído e a nível de precisão.

Gannotti et al. (2007) introduziram uma extensão ao paradigma de exploração de padrões sequenciais para a análise de objetos em movimento. Estes padrões estão relacionados com locais de interesse numa trajetória, explorando-se locais das mesmas, visitados sequencialmente, com o mesmo tempo de transição entre estes.

Lee et al. (2007) apresentaram uma *framework* de particionamento e agrupamento para o *clustering* de trajetórias. Esta abordagem começa por utilizar um algoritmo de particionamento nas trajetórias, criando um conjunto de sub-trajetórias. O processo de particionamento deverá garantir o máximo de precisão, sendo a diferença entre a trajetória e as sub-trajetórias mínima, e o máximo de concisão, minimizando o número de sub-trajetórias resultantes. Para atingir estes objetivos utilizou-se o princípio de M*inimum Description Length*. Depois, é aplicado um algoritmo de *clustering* de densidade sob os segmentos criados previamente. A função de distância definida tomará, como medida de similaridade entre os segmentos, a distância perpendicular, a distância paralela e a distância angular. O algoritmo de clustering é baseado no DBSCAN, mas, ao contrário deste, nem todos os conjuntos ligados por densidade dão, obrigatoriamente, origem a um *cluster*. Como condição para formarem um *cluster,* vários segmentos ligados por densidade não podem, por exemplo, pertencer à mesma trajetória.

Segundo Alvares et al. (2007), para se extraírem padrões relevantes, a semântica geográfica do local correspondente às trajetórias exploradas deve ser levada em consideração. É apresentada uma *framework* para explorar e modelar padrões de movimento sob o ponto de vista semântico. Desta forma pretende facilitar a compreensão dos dados permitindo a realização de *queries* aos mesmos, após a sua exploração, de uma forma que, por exemplo, a visualização de grupos de trajetórias, não permite.

Por outro lado, mais com o intuito de apoio à análise visual de um grande número de trajetórias, Rinzivillo et al. (2008) propuseram uma abordagem de *clustering* progressiva. É proposto um conjunto de funções de distância, em que cada uma

pretende explorar determinados atributos das trajetórias. Esta abordagem é progressiva porque permite que o resultado da utilização de uma determinada função de distância seja utilizado como entrada para outra. É dado um exemplo de uma exploração através de *clustering* de densidade sob dados de GPS de carros. No primeiro processamento utiliza-se um número mínimo de vizinhos elevado, resultando clusters de grande dimensão. Após a sua análise, os seus objetos deverão ser retirados do conjunto de dados. Efectuam-se mais processamentos necessários à compreensão dos dados, diminuindo-se o número mínimo de vizinhos para cada processamento.

Andrienko et al. (2009) introduz um conceito de *clustering,* também ele, em certa medida, progressivo, embora, neste caso, o processo seja iniciado com um pequena parte do conjunto de dados, formando-se *clusters* através de um algoritmo de densidade. Neste caso utilizou-se o algoritmo OPTICS. Depois vai-se adicionando cada objeto aos *clusters* criados inicialmente. São seleccionados pontos, denominados *"prototypes",* em cada *cluster*. A entrada de um ponto ainda não processado para um determinado *cluster* depende de se a sua distância a um *prototype* desse *cluster* é menor que limite predeterminado. Esta abordagem é mais adequada a uma grande quantidade de dados e deve ser gerida pelo analista durante o seu processamento.

Santos et al. (2012) propõe uma abordagem menos sustentada na orientação e aptidão do analista, mas mais automatizada. É descrito um processo de *clustering* de vetores de movimento em vez de se efetuar a construção das trajetórias previamente, através dos mesmos. Não há, portanto, necessidade de reconstruir as trajetórias no que toca à sua forma geométrica, a sua posição no espaço, mudanças de velocidade, direção, entre outros atributos ao longo do tempo.

Os autores utilizaram o algoritmo SNN devido à sua capacidade de identificar *clusters* com forma convexa e não convexa, tendo diferentes tamanhos e densidades, bem como por causa da sua capacidade para lidar com ruído. O número de *clusters* emerge diretamente dos dados, não tendo de ser introduzido pelo analista com base no seu conhecimento do domínio. A função de distância original deste algoritmo é baseada na distância euclidiana entre pontos, no entanto, esta função deverá, agora, lidar com propriedades próprias de um vetor de movimento. A função de distância foi alterada

para acomodar a posição e o *bearing* (direção) de um dado vetor. Foram também definidos e implementados diferentes pesos para cada uma destas variáveis, para que sejam identificados diferentes tipos de *clusters*.

Os resultados obtidos demonstraram que não é necessário reconstruir as trajetórias para que se possa identificar, no caso específico do trabalho apresentado, rotas de tráfego (Santos et al., 2012). Uma vantagem desta abordagem é que não é necessário conhecimento prévio para selecionar rotas ou regiões de interesse. O principal requisito é o ajuste dos parâmetros de entrada do algoritmo SNN, visto poderem influenciar as rotas de tráfego obtidas. No entanto é referido o custo computacional do processamento do algoritmo SNN, sugerindo que o pré-processamento de dados seria uma boa estratégia para limitar o número de vetores de movimento que seriam necessários para encontrar as rotas de tráfego.

Existe uma forte presença de algoritmos baseados em densidade de pontos na análise de dados sobre movimento. No entanto, a grande quantidade de dados deste tipo disponível hoje em dia para análise, torna o uso deste tipo de algoritmo cada vez mais problemático. Denota-se já algumas tentativas de aumentar a eficiência destes algoritmos. No entanto é uma um assunto que ainda carece de muito trabalho e desenvolvimento.

No próximo capítulo será abordada esta problemática, procurando-se desenvolver melhorias de desempenho no algoritmo SNN, no que respeita a tempo de processamento.

Capítulo 3
Otimização do Tempo de Processamento do *Clustering* via SNN

O algoritmo SNN é usado na análise de dados sobre movimento devido à sua capacidade para lidar com ruído nos dados e com *clusters* de várias formas. No entanto, este algoritmo apresenta uma grande dificuldade em lidar com grandes *datasets*. O tempo de processamento aumenta exponencialmente relativamente à quantidade de pontos a serem processados. Este problema vai ser analisado e serão apresentadas soluções para aumentar o desempenho neste sentido.

Neste capitulo é, em primeiro lugar, apresentada a metodologia seguida, de forma a perceber o problema e avançar para soluções de uma forma informada. É também referido o *dataset* usado para testes e são definidos alguns princípios relativamente aos dados de teste e aos próprios testes. Numa segunda parte, é apresentada uma implementação do algoritmo SNN. Como foi referido na revisão de literatura, este algoritmo produz resultados muito relevantes, no entanto o custo de utilização é elevado devido à sua ineficiência.

Na segunda parte deste capítulo são apresentados desenvolvimentos ao próprio algoritmo SNN, de forma a reduzir o problema de ineficiência do mesmo. Cada etapa do trabalho de optimização do algoritmo será descrita e testada individualmente. Todas as

implementações necessárias foram efetuadas na linguagem java.

3.1 Metodologia

Em primeiro lugar será abordado o algoritmo SNN, implementado-se uma aplicação cujo propósito é testar, avaliar e identificar os problemas deste algoritmo, de forma a se definir uma estratégia de abordagem ao mesmo, com o objetivo de aumentar a sua eficiência.

Numa fase posterior serão definidas novas soluções, ou seja, serão concetualizadas e implementadas abordagens variantes do algoritmo SNN, das quais resultem os mesmos resultados de clustering do SNN original, com o objetivo claro de reduzir o tempo de processamento. Cada abordagem desenvolvida será avaliada comparativamente às restantes, e o seu desempenho analisado no que concerne o tempo de processamento.

Para os testes realizados quer à aplicação SNN, quer às novas abordagens, foi usado um *dataset* real, denominado por *delft*, com 264.386 pontos. Este *dataset foi* obtido através de atividades realizadas no exterior, recorrendo a equipamentos com GPS, registando-se as coordenadas geográficas a cada instante de tempo. O *dataset* possui, além das coordenadas geográficas, alguns atributos caraterísticos de dados sobre movimento, como velocidade instantânea, *bearing* (direção), e momento no tempo em que cada registo é realizado. O *dataset* delft não possuía *bearing* (direção) para cada ponto, que teve de ser calculado por meio de interpolação, através das coordenadas geográficas de dois pontos consecutivos. Esta interpolação de dados foi realizada para se perceber o efeito da inclusão de outras dimensões, além das coordenadas geográficas, na análise de dados sobre movimento. No caso dos testes realizados, para além das coordenadas geográficas, é também utilizado o *bearing* para calcular a distância entre dois pontos, tendo o bearing um determinado peso neste cálculo.

Os conjuntos de dados para teste com diferentes tamanhos provêm deste mesmo *dataset*, sendo extraídos desde o primeiro elemento até ao elemento na posição respeitante ao tamanho pretendido para a realização de cada teste.

Para efeitos de análise dos dados, foi também necessário calcular a distância entre os dois pontos mais afastados geograficamente. Este dado é importante para normalizar a distância euclidiana, e se poder combinar a mesma com outras dimensões (como o *bearing*), no cálculo de distâncias do algoritmo SNN.

Todos os resultados obtidos nos testes realizados têm de ser considerados levando em conta apenas este *dataset.* Estes não servem como medida de comparação com outros algoritmos, pois estes podem ter sido testados com um dataset diferente, ou num ambiente de desenvolvimento diferente. Támbém os parâmetros de entrada que é necessário fornecer antes de um processamento do SNN têm influência no tempo de processamento. Para todos os testes realizados, os parâmetros de entrada foram K=10, Eps=3, e MinPts=7, dado serem valores frequentemente usados noutros testes realizados e descritos na literatura analisada. Desta forma, os valores de tempo de processamento, para cada teste, servem apenas para estabelecer comparações entre si.

De seguida, será descrito o trabalho de construção de uma aplicação para a utilização do algoritmo SNN, efetuado inicialmente.

3.2 Implementação do Algoritmo SNN

Já existia uma implementação prévia codificada em “Mathematica”, onde a aplicação que irá ser aqui descrita se baseou, e com a qual foram comparados resultados de processamentos, de forma a assegurar a fiabilidade dos mesmos, com alguma segurança.

Antes de mais interessa descrever algumas considerações do ponto de vista da codificação do algoritmo. Este foi dividido em três partes principais:

1. Cálculo de K vizinhos mais próximos.
2. Definição de pontos *core.*
3. Construção dos *clusters*.

Para o processamento do SNN, o analista precisa primeiro definir os parâmetros de entrada K, Eps e MinPts. Como já foi referido anteriormente, para todos os testes

efetuados, K foi definido como 10, Eps como 3, e MinPts como 7.

No processamento do SNN,em primeiro lugar, são calculados os K pontos mais próximos de cada ponto. A proximidade entre dois pontos não tem de ser necessariamente do ponto de vista espacial, embora no *clustering* de dados sobre movimento, é comum a distância euclidiana entre dois pontos assumir um papel importante.

O SNN não implica, portanto, uma função de distância pré-estabelecida. Este é um fator que surte efeitos no tempo de processamento. Em princípio, se a função de distância utilizada for mais complexa, isso significará mais tempo de processamento. Visto ser necessário definir uma função de distância para ser utilizada igualmente em todos os processamentos de teste, para que este fator não influencie as comparações dos resultados, foi escolhido um único exemplo. Este utilizará mais do que as duas dimensões das coordenadas geográficas, usadas no cálculo da distância euclidiana. Também utilizará o *bearing. Desta* forma será possível, quando for pertinente, alterar o peso atribuído à distância euclidiana e ao *bearing.*

Função de Distância

Nos testes realizados serão utilizadas as coordenadas geográficas, calculando-se a distância euclidiana entre dois pontos, e o *bearing,* calculando-se a diferença desta dimensão entre dois pontos.

Para combinar as duas medidas referidas, distância euclidiana e bearing, é necessário normalizar o valor obtido de cada uma. Isto significa que cada uma das medidas obtidas terá de ser dividida pelo máximo possível para essa medida neste *dataset.*

Para apresentar a função de distância usada, serão ilustradas as fórmulas necessárias, mas antes passa-se a especificar algumas variáveis usadas nas mesmas:

- **Wp** – Peso atribuído ao resultado da distância euclidiana. O peso atribuído à distância euclidiana vai definir a sua importância para o resultado final da distância entre dois pontos. O peso poderá ir de 0% a 100%. O peso para a

distância euclidiana utilizado em todos os testes do algoritmo será de 95%, dado ser um valor geralmente usado na descoberta de padrões de movimento.

- **Wb** – Peso atribuído ao resultado da diferença de *bearings*. Da mesma forma que *Wp* define a importância da distância euclidiana na distância final, *Wb* define a importancia da diferença de *bearing* para o resultado final da distância. O peso para a diferença de *bearing* utilizado em todos os testes do algoritmo será de 5%.

- **maxD** – Distância euclidiana máxima encontrada no *dataset* entre qualquer par de pontos. Este valor deve ser atualizado sempre que se explora um novo *dataset,* visto ser uma caraterística específica do mesmo.

- **maxB** – Diferença de *bearing* máxima encontrada no *dataset.* Assume-se que esta variável é sempre 180 graus.

Adotou-se, para efeitos de exemplo, os pontos $p_1(X_1,Y_1,B_1)$ e $p_2(X_2,Y_2,B_2)$ em que X_1 e X_2 representam a dimensão longitude, ou coordenada no eixo *X*, Y_1 e Y_2 representam a dimensão latitude, ou coordenada no eixo *Y*, e tanto B_1 como B_2 representam a dimensão *bearing* dos pontos p_1 e p_2 respetivamente. A primeira fórmula utilizada, denominada por *beringDif*, deverá retornar a diferença de *bearing,* em graus, através das dimensões B_1 e B_2.

$$\text{bearingDif}(p_1, p_2) = \textit{if}\ (|B_1 - B_2| > 180)\ \textit{then}\ (360 - |B_1 - B_2|)\ \textit{else}\ (|B_1 - B_2|)$$

A fórmula *bearingDif* será utilizada numa fórmula mais geral de distância. Esta sim, trata a combinação das dimensões utilizadas, utilizando-se as variáveis descritas anteriormente.

$$\text{distância}(p_1, p_2) = \text{Wp} \times \frac{\sqrt{((X_1 - X_2)^2 + (Y_1 - Y_2)^2)}}{maxD} + \text{Wb} \times \frac{\text{bearingDif}\,(p_{1,} p_2)}{maxB}$$

Utilizando a função de distância apresentada, cada ponto será comparado com todos os outros presentes no *dataset.* Desta forma, para cada ponto, é constituída uma lista com os K pontos que lhe são mais próximos.

Numa segunda fase de implementação do SNN, trata-se da definição de pontos *core* no *dataset.* Este processo envolve que, em primeiro lugar, se compare cada ponto com os que fazem parte da sua lista de K vizinhos, no sentido de perceber com quais partilha mais de Eps vizinhos. Se dois pontos se tiverem mutuamente nas suas listas de K vizinhos, e partilharem mais de Eps vizinhos da sua lista de K mais próximos, são considerados pontos reflexivos. A relação de reflexividade entre dois pontos é importante porque um ponto será considerado *core* se for reflexivo com *MinPts* ou mais dos seus K vizinhos.

Após encontrar os pontos *core* no *dataset*, passa-se à construção dos *clusters*. Enquanto existir algum ponto *core* sem *cluster,* é criado um novo, atribuíndo esse ponto *core* ao novo cluster.

Cada *cluster* começa, então, por ter apenas um ponto *core*. O *cluster* é depois construído através das relações de reflexividade, a partir desse ponto *core.* Isto significa que os pontos reflexivos do primeiro ponto do cluster também são adicionados ao *cluster.* Para esse novo conjunto de pontos adicionados ao *cluster*, são selecionados apenas os que também são pontos *core* e serão processados da mesma forma que o primeiro ponto *core* do cluster, isto é, os seus pontos reflexivos também serão adicionados ao *cluster*, sendo este processo executado para qualquer novo ponto *core* que entra para o *cluster.* Todos os pontos que entraram no *cluster* por serem reflexivos de um ponto *core* já pertencente ao *cluster,* mas não são eles próprios *core,* passam a ser considerados pontos de fronteira do *cluster*, ou seja, pontos *border.*

Quando já não existe mais nenhum ponto que partilhe uma relação de reflexividade com um ponto *core* do cluster em formação, que ainda não lhe pertença, o cluster pode ser considerado completamente formado. Repete-se o processo de formação de um *cluster* a partir de qualquer outro ponto *core* que ainda não pertença a nenhum. O processo de *clustering* pode considerar-se terminado quando não existir nenhum ponto *core* que não pertença a nenhum *cluster.*

Este algoritmo foi implementado em java, já com alguns cuidados tendo em vista a otimização da eficiência da aplicação, e com algumas considerações no que toca a interface e configurações, de forma a que possa ser utilizada facilmente em exploração de dados sobre movimento.

De seguida é apresentada a interface principal da aplicação, na Figura 3.1. Esta serve para especificar os parâmetros de entrada do algoritmo SNN, bem como os seus dados de entrada, e local e formato dos dados de saída. Os dados de entrada, como os dados gerados, podem ser armazenados em ficheiros CSV ou em bases de dados MySQL ou PostgreSQL. Os campos são preenchidos no inicio, a partir de um ficheiro de configuração, que grava o seu estado na ultima utilização.

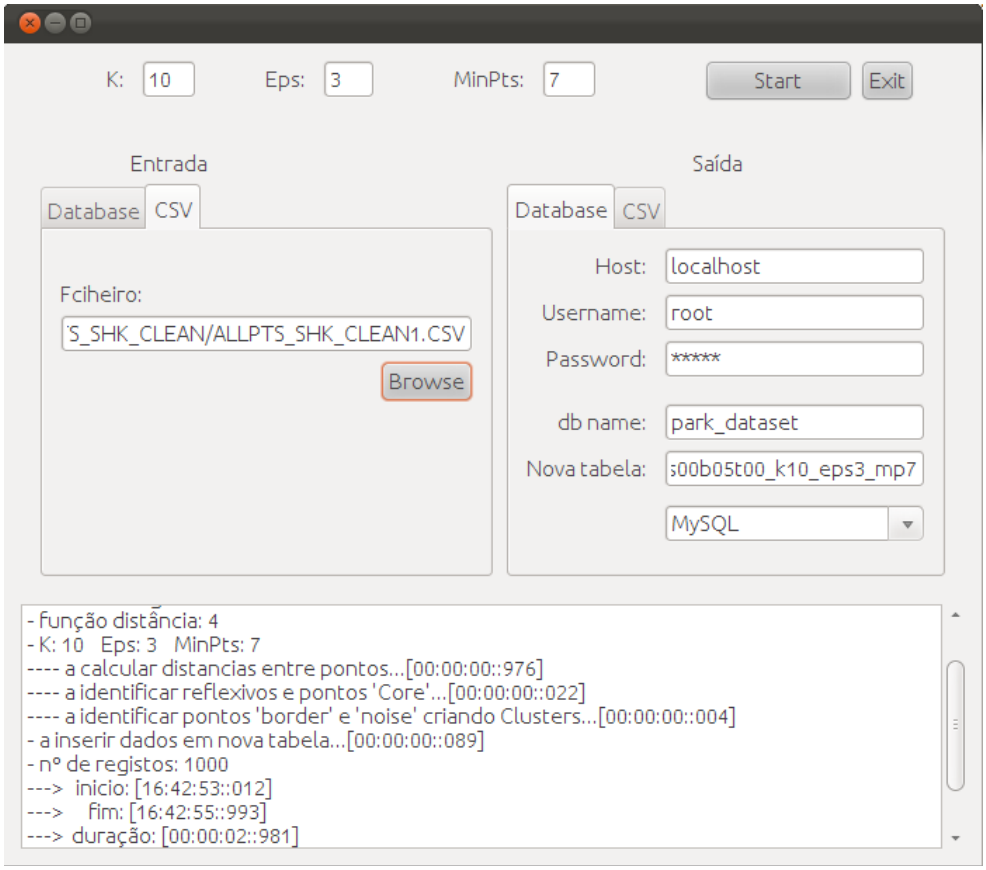


Figura 3.1: Interface gráfico da implementação do algoritmo SNN

Deu-se assim inicio à abordagem ao problema do *clustering* de grandes quantidades de dados sobre movimento. De seguida são apresentados os resultados obtidos de testes de desempenho, bem como algumas considerações face a estes

resultados.

3.2.1 Resultados Obtidos

A realização de testes, até esta fase, teve por principal objetivo validar a correção dos resultados da implementação efetuada, averiguando se são exatamente iguais ao resultado obtido através de uma implementação pré-existente, e perceber a complexidade de execução do algoritmo com as otimizações já integradas.

O gráfico da Figura 3.2 mostra os tempos de execução, em milissegundos, das fases do algoritmo referidas na descrição do SNN feita anteriormente.

Para o conjunto de dados analisado, utilizando como função de similaridade a distância euclidiana, o cálculo de distâncias e sua ordenação nas listas de K pontos de cada ponto, será a operação mais restritiva à eficiência do algoritmo.

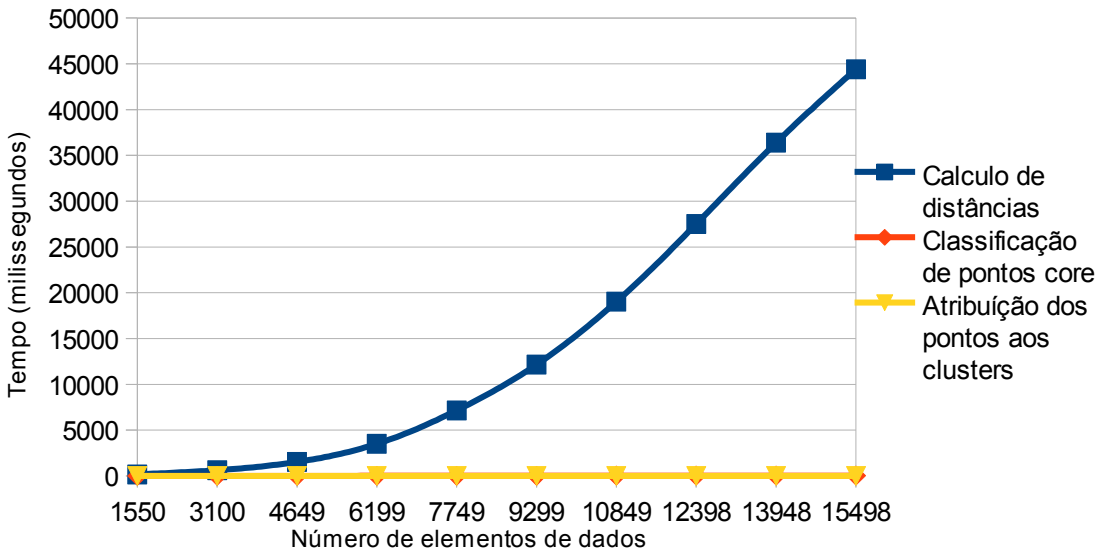


Figura 3.2: Fases do algoritmo e o seu tempo de execução.

Como já foi abordado na descrição da classe SNN do algoritmo, o cálculo de distâncias tem uma complexidade de execução de $n(n-1)/2$, logo, quanto maior for a quantidade de dados, mais marginal será a duração das restantes etapas do SNN

relativamente à duração global.

Usando ainda a mesma função de similaridade, na máquina utilizada para todos os testes realizados, uma operação de cálculo de distância e ordenação de cada um dos dois pontos do par, nos K vizinhos mais próximos do outro, demora cerca de 0.00035 milissegundos.

Através da fórmula *0.00035*n(n-1)/2,* sendo *n* o número de elementos de dados, é então possível calcular o tempo esperado de processamento para uma determinada quantidade de dados. De seguida, no gráfico da Figura 3.3, são comparados os tempos de execução reais com os esperados.

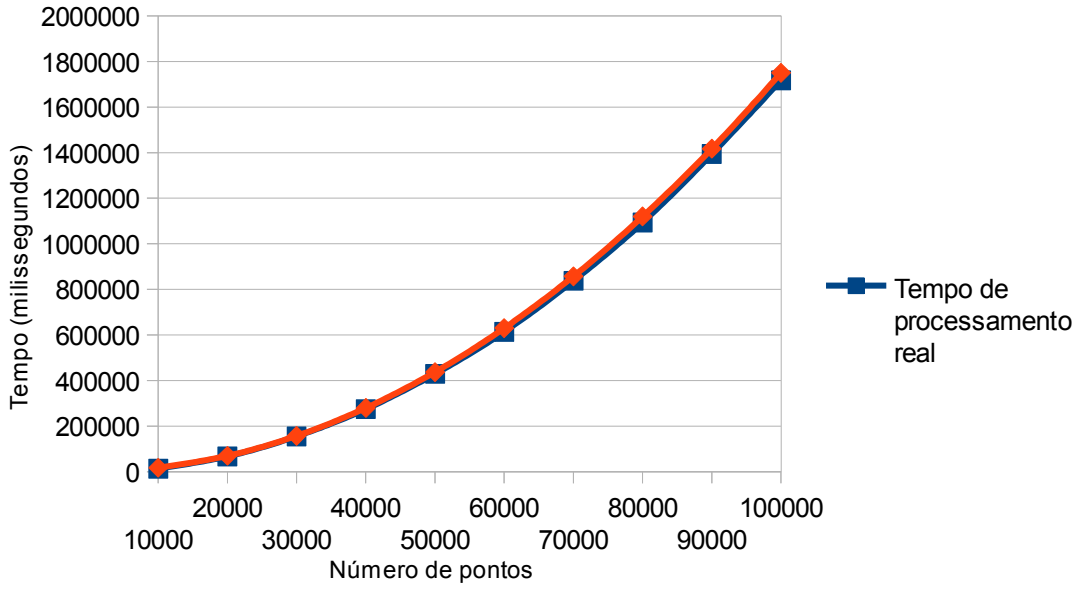


Figura 3.3: Tempo de processamento real e esperado da implementação do algoritmo SNN.

É possível verificar a elevada aproximação dos tempos esperados aos tempos reais de processamento. Como o tempo de processamento real engloba todas as etapas do algoritmo SNN, fica provado que é a fase de cálculo dos K vizinhos que tem maior impacto no tempo de processamento, e que limita o número de pontos a processar.

De seguida será descrito o esforço de melhoria de desempenho deste algoritmo, usando o conhecimento adquirido na exploração desta implementação do algoritmo SNN.

3.3 Desenvolvimento de Novas Abordagens ao Algoritmo SNN

A análise de dados sobre movimento é fortemente influênciada pela dimensão espacial associada aos dados. Isto é, esta análise leva fortemente em consideração a posição no espaço dos elementos de dados. Partindo deste pressuposto, fará sentido limitar o espaço de comparação entre elementos de dados com base nas suas coordenadas espaciais, princípio seguido, por exemplo, por Bhavsar e Jivani (2009).

Nesta secção serão descritas algumas abordagens principais propostas neste trabalho, resultantes do processo decorrido para minimizar o problema de eficiência do algoritmo SNN, com principal foco no cálculo de vizinhos mais próximos, que é notoriamente a fase mais demorada, como já foi verificado anteriormente.

Estas abordagens vão pretender tirar partido do uso comum da distância euclidiana para definir a proximidade entre pontos. O problema do SNN é a necessidade de calcular a proximidade de cada ponto do *dataset* com todos os outros pontos. A ideia base que se pretende explorar é reduzir significativamente o número de pontos a que cada ponto vai ser comparado. Para isso, os pontos vão ser distribuídos por uma matriz, de acordo com a sua coordenada *X* ou *Y.* Uma célula dessa matriz será referida como "secção". Como se vai procurar as distâncias euclidianas mais pequenas para cada ponto, será apenas necessário calcular a distância euclidiana de um determinado ponto com os pontos mais próximos de si no espaço, ou seja, cada ponto apenas precisa de ser comparado com os restantes que se encontram na mesma secção, e nas secções diretamente adjacentes à sua.

Este conceito é apresentado na Figura 3.4, onde para o ponto *P,* os 10 pontos mais próximos segundo a distância euclidiana (de cor vermelho escuro), se encontram na sua secção ou nas secções diretamente adjacentes (a amarelo) à secção em processamento (a verde).

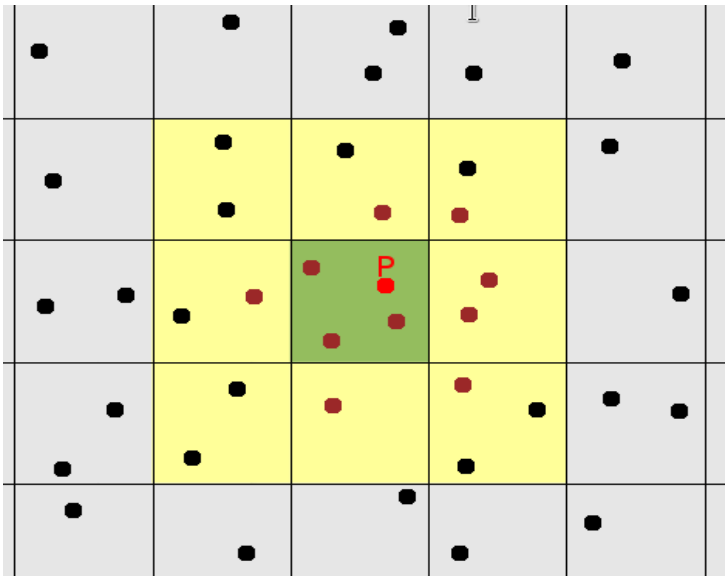


Figura 3.4: Conceito da distribuição dos pontos por uma matriz.

As ideias de cada abordagem, que a seguir se apresentam, foram implementadas em *java* e testadas com os dados sobre movimento presentes no *dataset* já referido anteriormente, com os mesmos parâmetros de entrada e função de distância. Foi também sempre utilizada a mesma máquina em condições semelhantes.

3.3.1 Abordagem 1: Divisão em Matriz da Dimensão Espacial

Numa primeira abordagem, a ideia será dividir os pontos por secções de espaço e comparar os pontos de cada secção com os restantes dessa secção, bem como com os pontos das secções anexas.

Em primeiro lugar é necessário tratar a divisão dos pontos no espaço. Para isso foi necessário definir um valor inicial correspondente ao número de divisões que cada dimensão espacial vai sofrer. Chamar-se-á a este valor a variável *M*. Para efetuar a divisão por *M* falta identificar a coordenada *X* máxima e mínima, assim como os mesmo valores para *Y*. Com estes dados obtém-se o valor dos limites de cada célula da matriz que irá ser utilizada para compartimentar o espaço. Os elementos de dados (pontos) são

então distribuídos pelas respectivas secções, ou células, da matriz e estão prontos para serem comparados quanto à sua similaridade.

O processamento dos vizinhos mais próximos de cada ponto é realizado por secção, uma de cada vez. Primeiro são comparados os vizinhos de uma secção entre si e, para cada um, é guardada a lista de K vizinhos mais próximos. Depois, para a mesma secção, os seus pontos são comparados com os pontos de todas as secções adjacentes, sendo que apenas as listas de vizinhos mais próximos dos pontos da secção original podem sofrer alterações. Ou seja, após o cálculo de uma distância entre um ponto da secção em processamento com um ponto de uma secção adjacente, apenas a lista de vizinhos mais próximos do primeiro ponto poderá incluir o segundo, e não o contrário. Assim evita-se a repetição de vizinhos na mesma lista, visto que cada ponto terá apenas uma oportunidade de entrar para a lista de vizinhos mais próximos de outro.

Esta ideia está ilustrada na Figura 3.5, que representa a matriz criada inicialmente. A secção representada a verde representa a secção em processamento num determinado momento, enquanto as secções sombreadas a amarelo representam as secções adjacentes a esta. As secções a cinzento mais escuro são secções processadas previamente, sendo a ordem, primeiro, de cima para baixo, e depois, da esquerda para a direita.

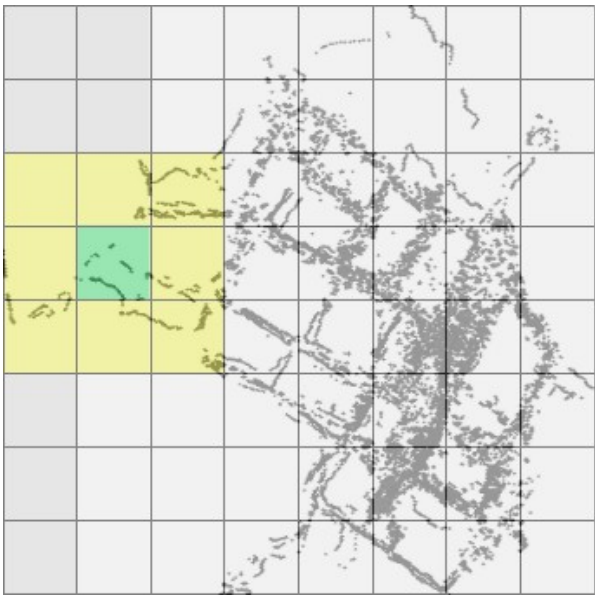


Figura 3.5: Representação das secções a processar num dado instante.

Partindo do princípio que os pontos de um determinado *dataset* em análise não estão concentrados em conjuntos muito densos, na dimensão espacial, quanto maior for o número de partições, menor será o número de cálculos de distâncias para cada ponto, de forma a se calcular a sua lista de vizinhos mais próximos. No entanto, a divisão dos pontos por um maior número de secções pode levar à ocorrência de erros na formação de *clusters*, relativamente aos resultados obtidos com o algoritmo SNN, devido à existência de dimensões na função de distância, que não podem ser reacionadas com as secções da matriz, como é exemplo, neste caso, o *bearing*.

Esta influência nos resultados que é originada pelo valor do parâmetro de entrada *M,* está relacionada com o não cálculo de distância entre dois pontos em que pelo menos um deles pertenceria à lista de vizinhos mais próximos de outro. Desta forma, a classificação de um destes pontos poderá sofrer alterações, o que poderá, por sua vez, influenciar os resultados do *clustering*.

Vejamos o caso em que dois pontos vizinhos se encontram em secções separadas por uma terceira. Nenhum dos pontos seria comparado com o outro, fazendo com que nenhum pudesse entrar para a lista de vizinhos do outro. Isto será mais comum

acontecer em zonas de baixa densidade de pontos ou em casos em que a dimensão espacial não é a única a ser levada em conta no cálculo da similaridade entre pontos.

Como já foi referido anteriormente, o cálculo do algoritmo SNN permite o cálculo de similaridade (ou distância) utilizando diferentes dimensões dos elementos de dados, neste caso pontos, dado que possuem dimensão espacial.

Introduzindo a noção de secção, os dados estão a ser limitados apenas nas suas dimensões espaciais, o que, só por si, poderá também gerar incorrecções nos *clusters* gerados. Pode ser referido, como exemplo, o caso do *bearing* ser levado em consideração no cálculo de similaridade. Se dois pontos possuírem *bearing* idêntico, mas estiverem um pouco afastados no espaço, eles podem ser separados por uma terceira secção mas serem suficiente similares para que pelo menos um deles pertencesse à lista de vizinhos mais próximos do outro. Este perigo aumenta quanto menor for o peso da dimensão espacial no cálculo de similaridade.

Caberá ao analista definir *M* de forma a minimizar a possibilidade de erro, mas tirando partido da compartimentação dos dados para aumentar o desempenho do algoritmo. Para isso será conveniente que o analista conheça o *dataset* e tenha alguma experiência prévia no nível de erro que a compartimentação pode introduzir.

De seguida são demonstradas as linhas guia da implementação desta abordagem.

Algoritmo da abordagem 1

```
matriz[][] ← Distribuir("lista de pontos")

for i=0 até matriz.length do // percorre colunas

    for j=0 até matriz[i].length do // percorre células da coluna

        pontos = matriz[i][j]

        pts_a_processar.addAll(matriz[i][j],matriz[i-1][j],

                               matriz[i-1][j-1],matriz[i-1][j+1],

            matriz[i][j-1],matriz[i][j+1],

                               matriz[i+1][j-1],matriz[i+1][j],

                                   matriz[i+1][j+1])

        calculaDistancia(pontos, pts_a_processar) // calcula as distâncias entre
cada ponto presente em "pontos" e os pontos a processar, presentes em
"pts_a_processar", ordenando estes últimos na lista de K vizinhos mais próximos do ponto.

    end for

end for
```

Resultados

Esta algoritmo foi implementado na linguagem *java,* usando muita da estrutura e classes criadas para a implementação original do algoritmo SNN referida anteriormente.

Após a implementação deste algoritmo, este foi testado para se perceber a influência deste conceito no desempenho do *clustering*. A Figura 3.6 mostra um gráfico comparativo do tempo de processamento entre este algoritmo e o SNN original, para diferentes quantidades de dados. Os *clusters* resultantes de ambos os processos são exactamente iguais.

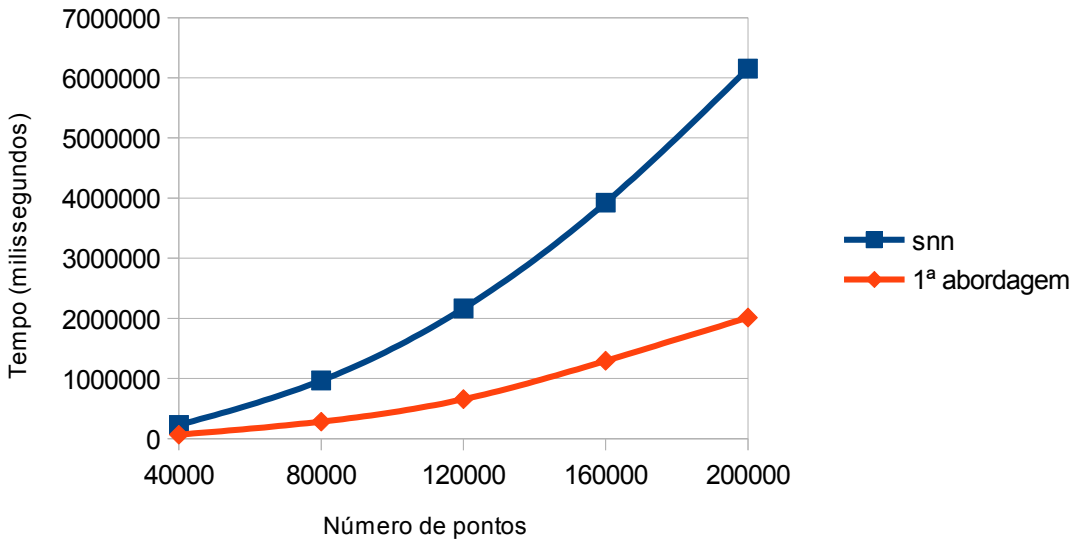


Figura 3.6: Tempo de processamento para o SNN e a abordagem 1.

Para o *dataset* com 200.000 pontos, os tempos de processamento registados são, aproximadamente, 1h42m para o SNN e 0h33m para o algoritmo resultante desta etapa. É notório maior desempenho, encorajando o desenvolvimento baseado nesta estratégia.

3.3.2 Abordagem 2: Aproveitamento de Cálculo Efetuado

Esta etapa pretendeu aproveitar o fato de que, quando uma distância entre dois pontos é calculada, esta tanto poderá ser utilizada para ordenar o primeiro ponto na lista vizinhos mais próximos do segundo, como vice-versa.

Esta abordagem é muito similar à anterior, mas, no passo de calcular distâncias entre pontos de uma secção com os pontos das secções adjacentes, pretende-se que os pontos das secções adjacentes possam aproveitar o cálculo efetuado para atualizarem a sua lista de vizinhos mais próximos.

O principal desafio nesta ideia é não haver cálculo de distância entre dois pontos mais do que uma vez. Assim sendo, passou-se a comparar os pontos de cada secção não com os pontos de todas as secções adjacentes, mas apenas com as secções que ainda não foram processadas. Agora é possível utilizar uma distância entre dois pontos, calculada apenas uma vez, para ordenar cada um na lista de vizinhos mais próximos do outro.

Na Figura 3.7 está representado um passo do processamento segundo esta abordagem. A secção representada a verde representa a secção em processamento num determinado momento, enquanto as secções sombreadas a amarelo representam as secções adjacentes a esta.

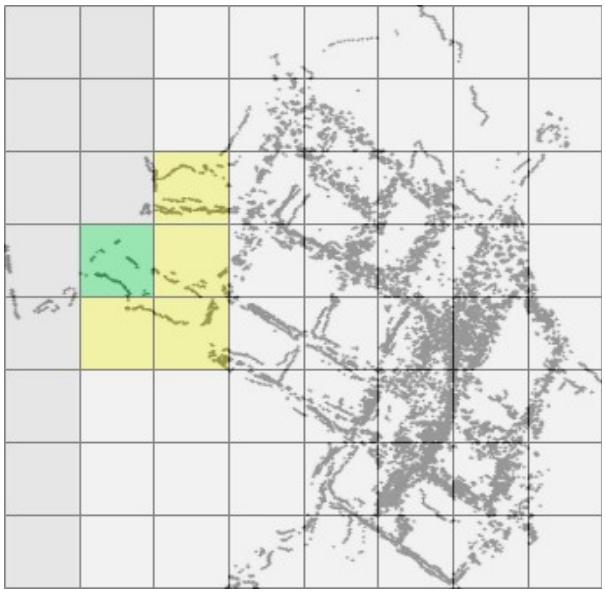


Figura 3.7: Representação das secções que serão processadas.

Nesta abordagem persistem os problemas relativos à probabilidade de ocorrência de erros se *M* for demasiado grande para o *dataset* em análise.

De seguida são demonstradas as linhas guia da implementação desta abordagem.

Algoritmo da abordagem 2

matriz[][] ← Distribuir(“lista de pontos”)

for i=0 até matriz.length ***do //*** percorre colunas

for j=0 até matriz[i].length ***do //*** percorre células da coluna

pontos = matriz[i][j]

pts_a_processar.addAll(matriz[i][j],matriz[i][j+1],

matriz[i+1][j-1],matriz[i+1][j],

matriz[i+1][j+1])

calculaDistancia(pontos, pts_a_processar) // Calcula as distâncias entre o cada elemento de *pontos* e os pontos a processar presentes em pts_a_processar, ordenando estes últimos na lista de K vizinhos mais próximos do ponto, mas agora, também ordenando o ponto de *pontos* na lista de K vizinhos mais próximos de cada ponto da lista “pts_a_processar”.

end for

end for

Resultados

Esta alteração foi efetuada sob a implementação da abordagem anterior. Também após a implementação deste algoritmo foram realizados alguns testes para se perceber a influência deste conceito no desempenho do *clustering*.

A Figura 3.8 mostra um gráfico comparativo do tempo de processamento, para diferentes quantidades de dados, entre esta abordagem, a abordagem 1, e o SNN original. O valor de *M* é 10, dividindo a dimensão espacial numa matriz 10*10.

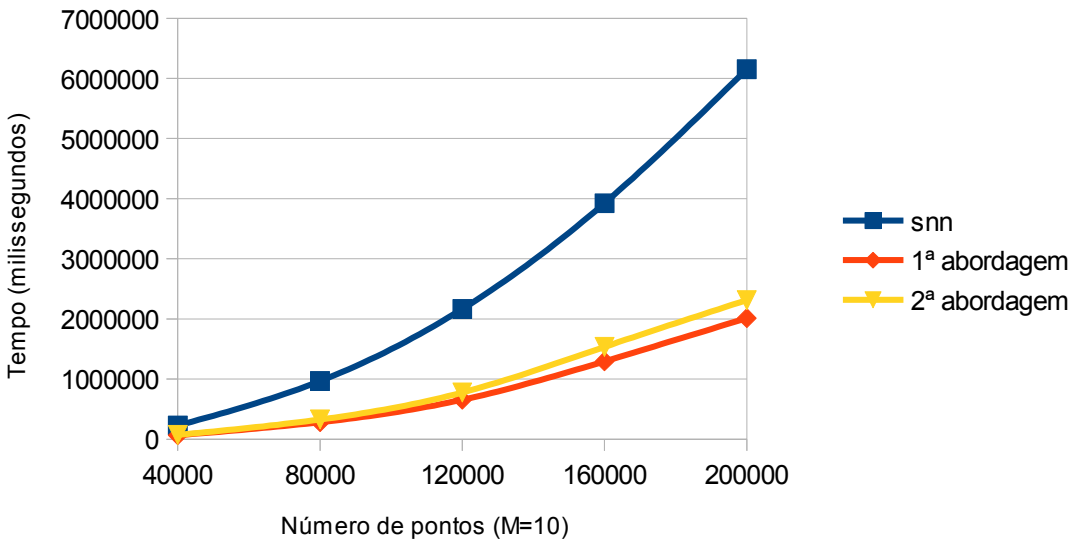


Figura 3.8: Tempo de processamento entre o SNN e as abordagens 1 e 2.

Para o *dataset* com 200.000 pontos, o tempo de processamento registado é, aproximadamente, 0h38m para o algoritmo resultante desta etapa. A eficiência diminuiu relativamente à abordagem 1, que tinha demorado 0h33 para processar os mesmos 200.000 pontos. Por esta razão, não se progrediu utilizando esta estratégia.

3.3.3 Abordagem 3: Expansão Iterativa

Nesta abordagem tenta-se solucionar um dos problemas identificados nas etapas anteriores. A dimensão espacial de dados sobre movimento possui, geralmente, zonas de muito grande densidade de pontos, e outras zonas de relativamente baixa densidade. Propõe-se que para cada secção da matriz, os pontos sejam comparados não só com os que estão presentes nas secções adjacentes, mas também com os pontos das secções seguintes iterativamente, tal como é ilustrado na Figura 3.9. A ideia principal é aumentar o valor de M, diminuindo o tamanho das células, sem que isso conduza a erros no cálculo dos K vizinhos mais próximos de cada ponto.

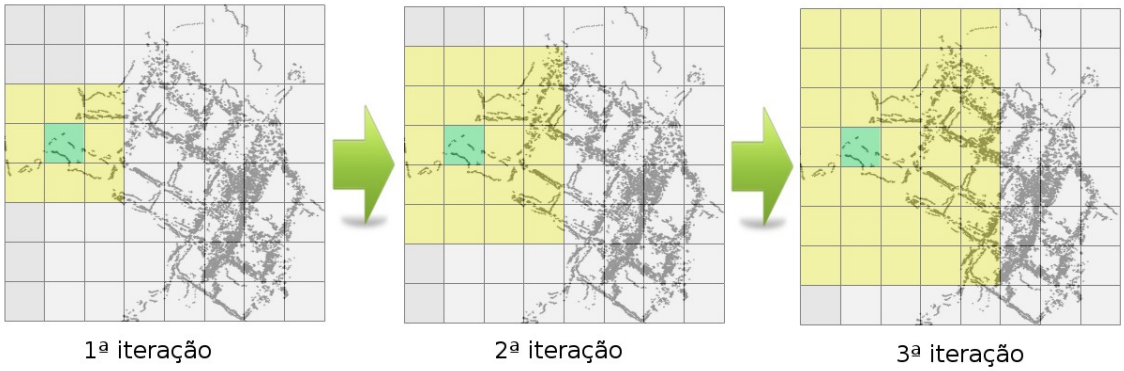


Figura 3.9: Iterações relativas ao processo de expansão.

As iterações deixam de ocorrer, para cada secção, quando nenhum ponto testado na última iteração tiver uma distância a qualquer ponto da secção em processamento suficientemente pequena para entrar na sua lista de vizinhos mais próximos, e todos os pontos da secção em processamento tenham pelo menos *K* vizinhos mais próximos na sua lista. Esta condição de paragem, para o processamento de cada secção, implica que o processamento de distâncias da última iteração não tenha, portanto, utilidade na descoberta de vizinhos mais próximos.

Esta abordagem permite garantir que, mesmo nas zonas de menor densidade de pontos no plano espacial, todos os pontos encontram os seus *K* pontos vizinhos mais próximos. Desta forma, é possível aumentar a variável *M* relativamente às abordagens anteriores, diminuindo-se assim a média de pontos que são comparados com cada um,

para se encontrar os seus *K* vizinhos mais próximos.

No entanto, persiste o erro derivado do efeito de outras dimensões não espaciais, no resultado de similaridade entre os pontos. Persiste também outra fonte de erro. Trata-se do facto das expansões se efetuarem sob a forma retangular. Isto significa que poderá ocorrer o lapso da condição de paragem ocorrer para uma determinada secção, mas existirem pontos relevantes para os resultados numa outra secção mais próxima da parte central de um dos lados do retângulo. Este problema encontra-se ilustrado na Figura 3.10, onde o ponto B, embora não tenha sido processado, seja mais próximo ao ponto A que os pontos encontrados antes da condição de paragem, como por exemplo o ponto C.

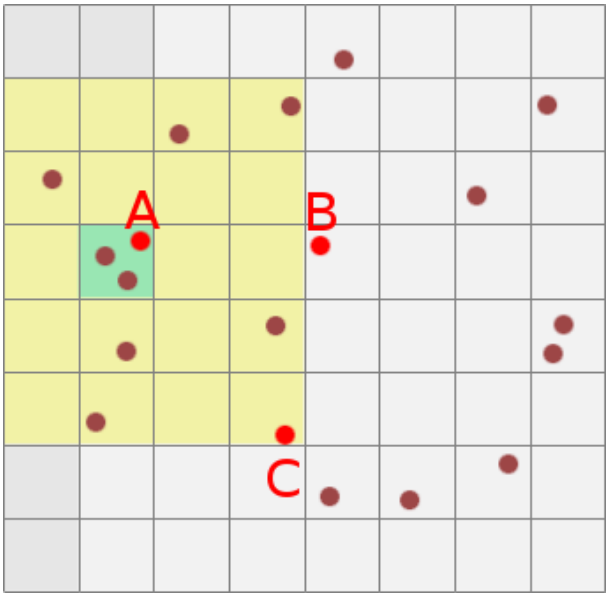


Figura 3.10: Erro derivado da expansão se efetuar em quadrado.

De seguida são demonstradas as linhas guia da implementação desta abordagem.

Algoritmo da abordagem 3

```
matriz[][] ← Distribuir("lista de pontos")

for i=0 até matriz.length do // percorre colunas

    for j=0 até matriz[i].length do // percorre células da coluna

        pontos = matriz[i][j]

        boolean cond = true

        nível = 1

        while (cond)

            pts_a_processar  =expansão (nível)
```

// O método *expansão* abstrai a extração dos pontos presentes nas células da matriz adjacentes. O *nível* representa a iteração da expansão a realizar.

```
            cond = calculaDistancia(pontos, pts_a_processar)
```

// *calculaDistancia* funcionará como na abordagem 1, mas agora devolve um booleano falso quando nenhum ponto da lista *pts_a_processar* entrar para a lista de K vizinhos mais próximos de qualquer ponto dos *pontos*. Caso contrário devolve um booleano verdadeiro.

```
            nível++

        end while

    end for

end for
```

Resultados

Esta alteração ao algoritmo foi efetuada sob a implementação da primeira abordagem. Interessa também, para este caso, verificar as diferenças entre o seu desempenho e as abordagens anteriores.

A Figura 3.11 mostra um gráfico comparativo do tempo de processamento, para diferentes quantidades de dados, entre este algoritmo, os apresentados anteriormente, e o SNN original. Dado esta abordagem ser menos propensa a erros, o valor de *M* poderá ser muito maior.

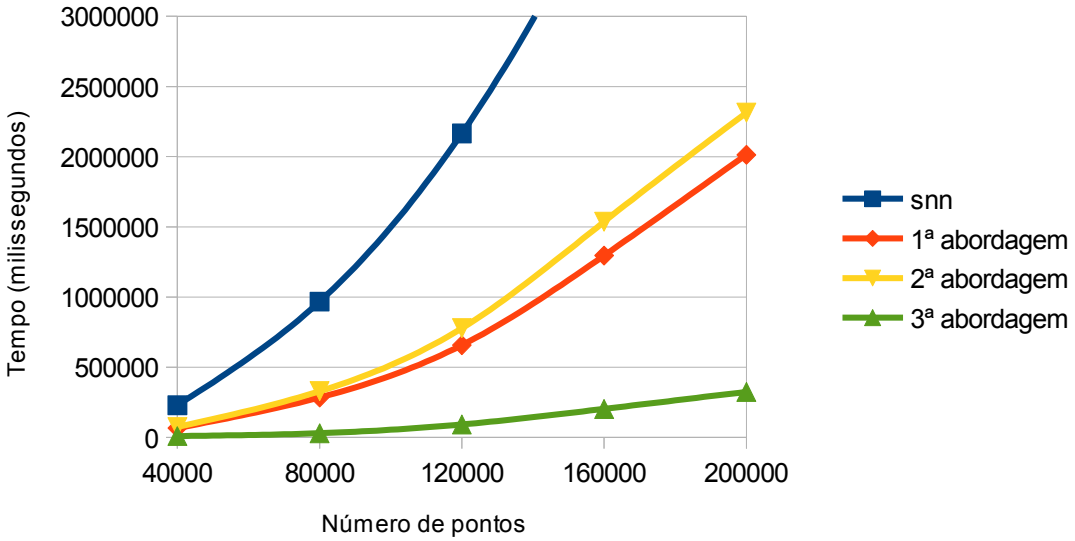


Figura 3.11: Tempo de processamento para o SNN e as abordagens 1, 2 e 3.

Para o *dataset* com 200.000 pontos, o tempo de processamento registado foi de, aproximadamente, 5m20s para o algoritmo resultante desta abordagem. O aumento de *M* mostra-se muito importante para melhorar o desempenho do algoritmo. No entanto aumentar mais o seu valor significaria um cada vez maior risco de provocar erros nos *clusters* resultantes do processamento. Para este *dataset*, *M*=50 parece ser um compromisso aceitável, mas, para diferentes *datasets,* com diferentes densidades de

pontos, este valor poderá ser aumentado, aumentando a eficiência, ou terá de ser diminuído para minimizar ou anular o erro. Assim sendo, não parece ser uma solução aceitável para a análise de dados sobre movimento, visto não ser possível conhecer o *M* mais indicado para cada *dataset* antecipadamente.

3.3.4 Abordagem 4: Expansão Iterativa Multi-Dimensional

Nesta etapa são abordados os restantes problemas, nomeadamente a influencia de outras dimensões, além da espacial, no resultado das distâncias entre pontos, e o problema da expansão ocorrer sob a forma retangular.

Também ela se baseando na anterior, esta abordagem passa a receber não *M* como parâmetro de entrada, mas antes *Mx*, significando o número e divisões realizado apenas no eixo horizontal da dimensão espacial. Assim, o *My* representará o número de divisões no eixo vertical, e será calculado automaticamente, de maneira que as expansões sejam feitas sob a forma de quadrados.

Mais eficaz para a redução da probabilidade de erro seria as expansões se efetuarem sob forma de círculo. Assim sendo, será empenhado algum de tempo de processamento neste sentido. Como é possível visualizar na Figura 3.12, quando se atinge a condição de paragem das expansões para uma determinada secção, vai-se continuar a expandir para cima, baixo e para os lados até se conseguir encaixar uma circunferência que toque nos vértices do quadrado desenhado na última expansão.

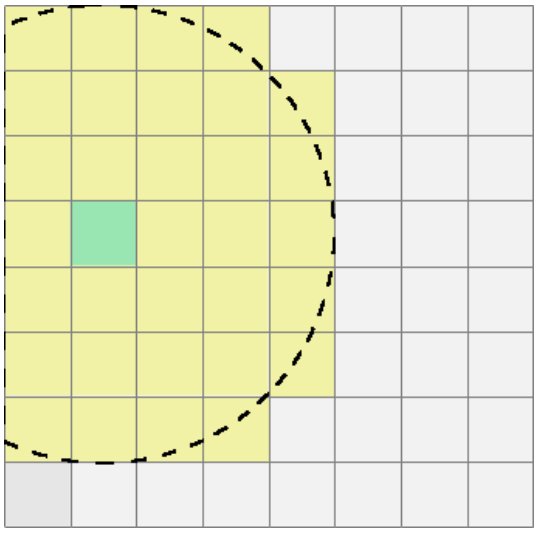


Figura 3.12: Expansão para abranger um círculo.

Para erradicar qualquer possibilidade de erro, independentemente da densidade de pontos, falta apenas tratar o efeito da utilização de outras dimensões no cálculo de distâncias entre pontos. Este fator foi tratado através do acréscimo de um número de expansões às efetuadas previamente, que dizem respeito ao peso das dimensões não espaciais no resultado da distância entre pontos.

Ainda antes do passo discutido anteriormente, relativo a realizar expansões para abranger uma circunferência, e após a expansão em que nenhum dos pontos sofre alterações na sua lista de K vizinhos, servindo por isso como condição de paragem, será efetuado um número de expansões calculado por uma função que será descrita de seguida.

Para determinar o número de expansões a realizar em cada secção em processamento, de forma a abranger o efeito de dimensões não espaciais nos dados, começa-se por encontrar o último ponto da lista de *K* vizinhos mais próximos em cada ponto da secção em processamento. Para cada um destes pontos encontrados, seleciona-se aquele que obteve maior distância ao ponto a cuja lista este pertence, tendo em conta apenas as dimensões presentes na função de distância excetuando as coordenadas espaciais. Esta distância máxima será denominada *b* para o caso do *bearing* .

Após calcular este valor máximo, é efetuada uma regra de três simples. O valor máximo possível que poderá ser atingido pela distância, medida apenas pelas dimensões presentes na função de distância excetuando as coordenadas espaciais, multiplicado pelo seu peso para o resultado da distância final (*Wb* para o caso do *bearing*), está para o número máximo de quadriculas seguidas *(m),* na vertical ou horizontal, também multiplicado pelo peso dado à dimensão espacial (*Wp*), como o valor máximo calculado anteriormente está para o *X.* Este valor *X* calculado deverá ser arredondado, sempre por excesso, às unidades. Agora, o valor *X* representa o número de expansões que deverão ser efetuadas para garantir que para nenhum ponto são negligenciados outros pontos que poderiam pertencer à sua lista de vizinhos mais próximos, devido a maior similaridade por parte das dimensões não espaciais.

Na implementação realizada para teste, a função de distância entre pontos, para além das dimensões das coordenadas espaciais, utilizou apenas o *bearing*, sendo então necessário calcular o número de expansões necessárias para que não ocorram erros no cálculo de listas de K vizinhos derivados desta dimensão. Para descrever a função que retorna o número de expansões a efetuar foram definidas as seguintes variáveis:

- ***b*** – Maior diferença de *bearing* encontrada entre um qualquer ponto da secção em processamento e o último ponto da sua lista de K vizinhos.
- ***m*** – Máximo(*Mx*,*My*).
- ***Wb*** – Percentagem de peso do *bearing* no cálculo de similaridade entre pontos.
- ***Wp*** – Percentagem de peso da posição espacial no cálculo de similaridade entre pontos.

O número de expansões a efetuar devido à dimensão *bearing* é dado pela seguinte função:

$$\text{Número de expansões} = \frac{\text{Wb} \times b}{180} \times \frac{m}{2 \times \text{Wp}}$$

Sintetizando, esta abordagem possui três fases principais. Em primeiro lugar são efetuadas expansões, calculando distâncias entre os pontos da secção em processamento

com pontos cada vez mais afastados no espaço. Quando for efetuada uma expansão em que nenhuma lista de K vizinhos dos pontos da secção em processamento sofra alterações, considera-se atingida uma condição de paragem. De seguida, numa segunda fase, é calculado o número de expansões a efetuar devido às dimensões diferentes das coordenadas espaciais presentes na função de distância, sendo depois efetuadas essas expansões, comparando todos os pontos obtidos com os presentes na secção em processamento. Finalmente, em terceiro lugar, são efetuadas expansões, agora sob a forma de cruz em vez de forma quadrada como as anteriores, até abranger uma circunferência.

Esta abordagem representa um passo muito importante no desenvolvimento de um algoritmo mais eficiente. É agora possível dar qualquer valor a *M* sem que isso influencie o resultado do processo de clustering. Como já foi referido anteriormente, aumentanto o valor desta variável prevê-se uma melhoria de desempenho muito significativa, principalmente em *datasets* com pontos muito concentrados en determinadas áreas do espaço.

De seguida são demonstradas as linhas guia da implementação desta abordagem.

Algoritmo da abordagem 4

```
matriz[][] ← Distribuir("lista de pontos")

for i=0 até matriz.length do // percorre colunas

    for j=0 até matriz[i].length do // percorre células da coluna

        pontos = matriz[i][j]

        boolean cond = true

        nível = 1

        while (cond)

            pts_a_processar  =expansão (nível)

            cond = calculaDistancia(pontos, pts_a_processar)

            nível++

        end while
```

nExp = número de expansões *(b)* *// nExp* representa o número de expansões necessárias para que não ocorram erros no cálculo de listas de K vizinhos derivados da dimensão *bearing*. Por sua vez, *b* representa a maior diferença de *bearing* encontrada entre um qualquer ponto da secção em processamento *(pontos)* e o último ponto da sua lista de K vizinhos.

```
        while (nExp>0)

            pts_a_processar  =expansão (nível)

            calculaDistancia(pontos, pts_a_processar)
```

nível++

nExp--

end while

circ = nExpCirc(nível) // *nExpCirc* recebe o nível atual de expansão e devolve o número de expansões necessárias para abranger um círculo

nível = 0

while (circ>0)

pts_a_processar =expansãoCirc (nível) // *expansãoCirc* representa o processo de expansão em cruz falado anteriormente.

calculaDistancia(pontos, pts_a_processar)

nível++

circ--

end while

end for

end for

Resultados

Esta alteração ao algoritmo foi efetuada sob a implementação resultante da aplicação das alterações discutidas na abordagem 3.

Embora exija algumas expansões extra que as abordagens anteriores, esta permite eliminar totalmente o erro derivado do seccionamento da dimensão espacial. Assim, o valor de *M* poderá atingir qualquer valor sem por isso ocorrer erro.

É por isso pertinente procurar o valor de *M* mais eficiente, face à densidade do *dataset* em questão. No gráfico da Figura 3.13 são mostrados os tempos de processamento, para o *dataset delft,* relativamente a diferentes valores de *M*, para diferentes pesos atribuídos à dimensão espacial. Como pode ser verificado nesse gráfico, a partir de *M*=300 a eficiência do algoritmo estabiliza, independentemente do peso atribuído à dimensão espacial.

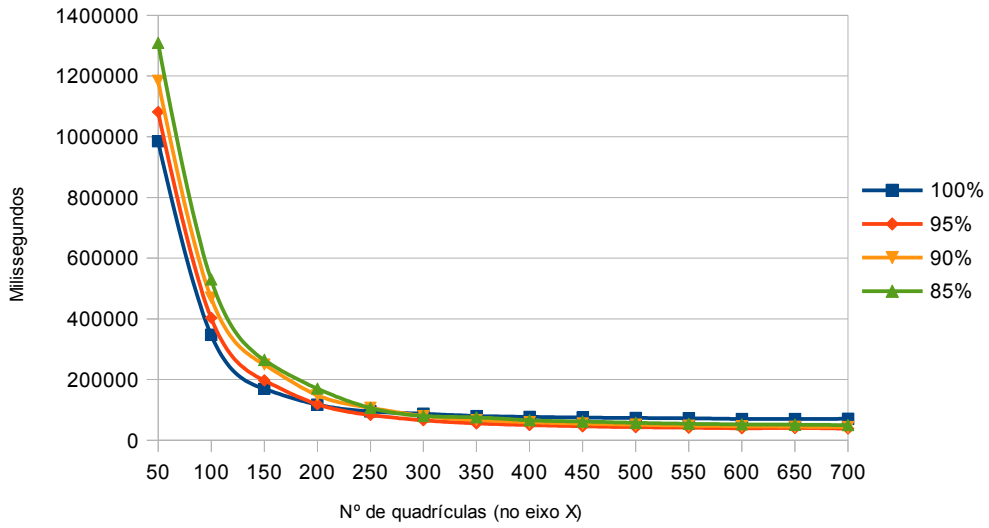


Figura 3.13: Tempo de processamento para diferentes pesos para a distância euclidiana.

Dada a ampla possibilidade de escolha para a variável *M*, adoptou-se o valor 500 por já ter uma margem aceitável para o caso de se utilizar um *dataset* com maior densidade.

É agora possível verificar as diferenças entre o seu desempenho e as abordagens anteriores. A Figura 3.14 mostra um gráfico comparativo do tempo de processamento, para diferentes quantidades de dados, entre este algoritmo, os apresentados anteriormente, e o SNN original. Agora a escala de tempo representada é mais curta, para que seja possível visualizar as diferenças.

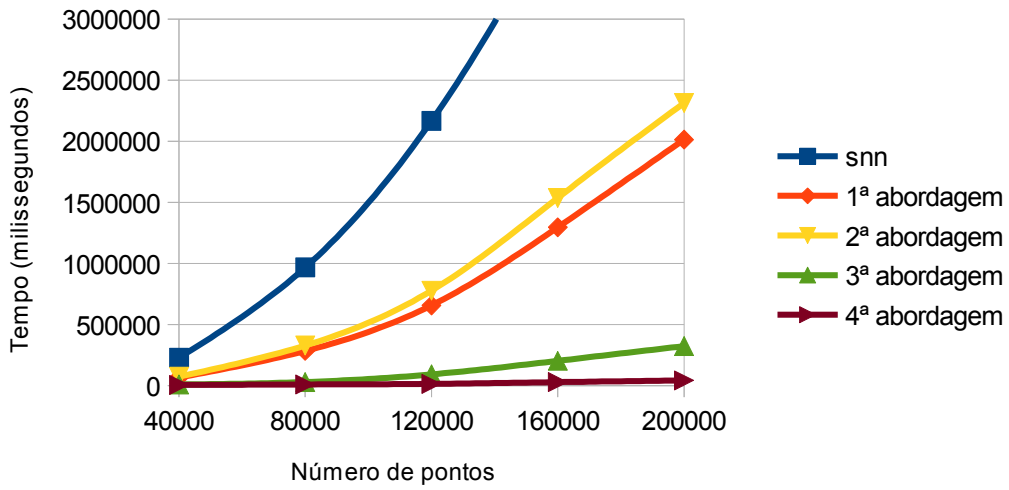


Figura 3.14: Tempo de processamento entre o SNN e as abordagens 1, 2, 3 e 4.

Através do gráfico da Figura 3.14 verifica-se que, para as quantidades de pontos testadas, este algoritmo demonstra uma curva de crescimento menos acentuada que nas etapas anteriores. Foram processados 200.000 pontos em cerca de 42 segundos, o que, comparado com 1h42m do SNN original, corresponde a um ganho muito considerável.

3.3.5 Abordagem 5: Expansão Iterativa Multi-Dimensional Independente

De forma a minimizar o custo de eficiência das expansões, pretendeu-se, nesta abordagem, passar a calcular independentemente as expansões a efectuar, para cada ponto presente na secção em processamento da matriz. Assim, à medida que pontos atingem a condição de paragem, são processados em expansões para levar em conta outras dimensões, e são ainda processados em expansões para abranger um círculo, estes são retirados do conjunto de pontos a ser processados nas expansões seguintes, para cada secção em processamento.

Seguindo esta estratégia, o custo de eficiência das últimas expansões deverá ir diminuindo na medida em que pontos vão sendo terminados, e excluídos do grupo em processamento. Ou seja, na generalidade dos casos, os pontos recolhidos com as últimas expansões não serão comparados com todos os pontos presentes na secção em processamento da matriz.

No entanto, isto implica ainda algum custo de desempenho. Quando a primeira condição de paragem é atingida por qualquer ponto, o número de expansões requeridas pelas dimensões não espaciais terá de ser calculado para esse ponto. Deste modo, o valor mencionado terá de ser calculado um número de vezes igual à quantidade de pontos existentes na secção em processamento.

Também o número de expansões necessárias para se abranger uma circunferência, terá de ser calculado uma vez para cada final de expansão, em que existam um ou mais pontos que transitem para esta fase.

De seguida são demonstradas as linhas guia da implementação desta abordagem.

Algoritmo da abordagem 5

matriz[][] ← Distribuir(“lista de pontos”)

for i=0 até matriz.length ***do //*** percorre colunas.

for j=0 até matriz[i].length ***do //*** percorre células da coluna.

pontos = matriz[i][j]

nível = 1

while (pontos.size > 0)

pts_a_processar =expansão (nível)

calculaDistancia(pontos, pts_a_processar) // agora, o restante processamento descrito para o algoritmo 4 é realizado dentro de cada ponto. Quando cada um chega à sua iteração final, retira-se da lista “*pontos*”.

nível++

end while

end for

end for

Resultados

Esta alteração ao algoritmo foi efetuada sob a implementação resultante da aplicação das alterações discutidas nas abordagens anteriores.

É agora possível verificar as diferenças entre o seu desempenho e as abordagens anteriores. A Figura 3.15 mostra um gráfico comparativo do tempo de processamento para diferentes quantidades de pontos entre este algoritmo, os apresentados anteriormente, e o SNN original.

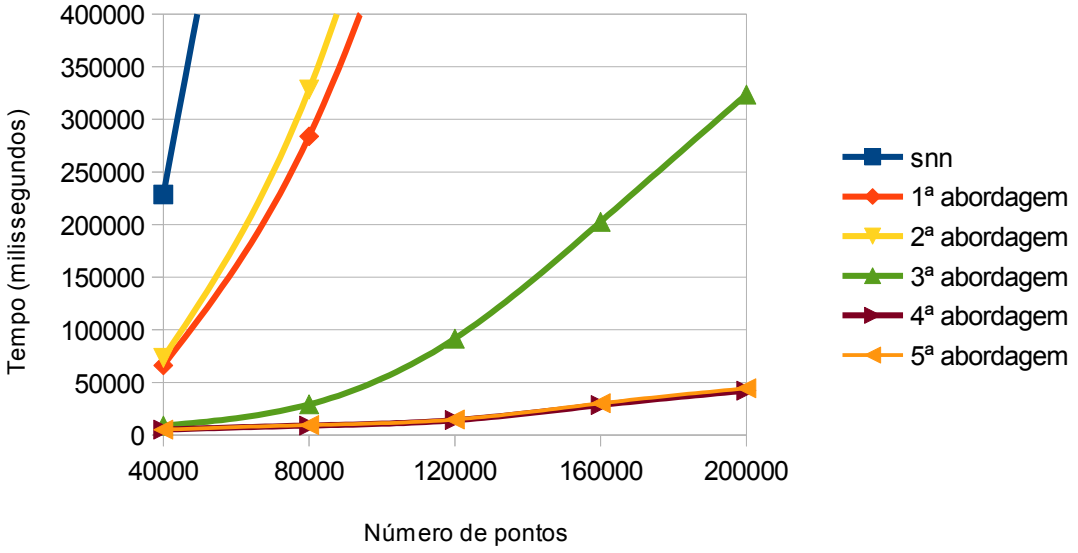


Figura 3.15: Tempo de processamento entre o SNN e as abordagens 1, 2, 3, 4 e 5.

Como é possível perceber a partir do gráfico da Figura 3.15, o desempenho deste último incremento ao algoritmo, aparentemente, não provoca ganho de eficiência. O tempo de processamento para 200.000 pontos é agora de 44 segundos, que não é melhor relativamente aos 42 segundo demorados na abordagem anterior, mas não deve ser esquecido que os testes realizados até ao momento foram sempre efetuados com um peso de 95% para a dimensão espacial.

Esta estratégia é pensada para ser mais eficiente, relativamente à anterior, quanto maior for o peso das dimensões para o cálculo da proximidade entre dois pontos. Este

pressuposto parte do princípio que, assim, o número de expansões necessárias para abranger as dimensões não espaciais variará de ponto para ponto. Deste modo, é possível excluir pontos da secção em processamento, que já tenham concluído o número de expansões que eram necessárias para si.

Na Figura 3.16 está ilustrado um gráfico comparando os tempos de processamento entre o algoritmo desta abordagem e da abordagem anterior, mas agora com um peso de apenas 50% para a distância euclidiana, e igualmente 50% para a diferença de *bearing* entre os pontos.

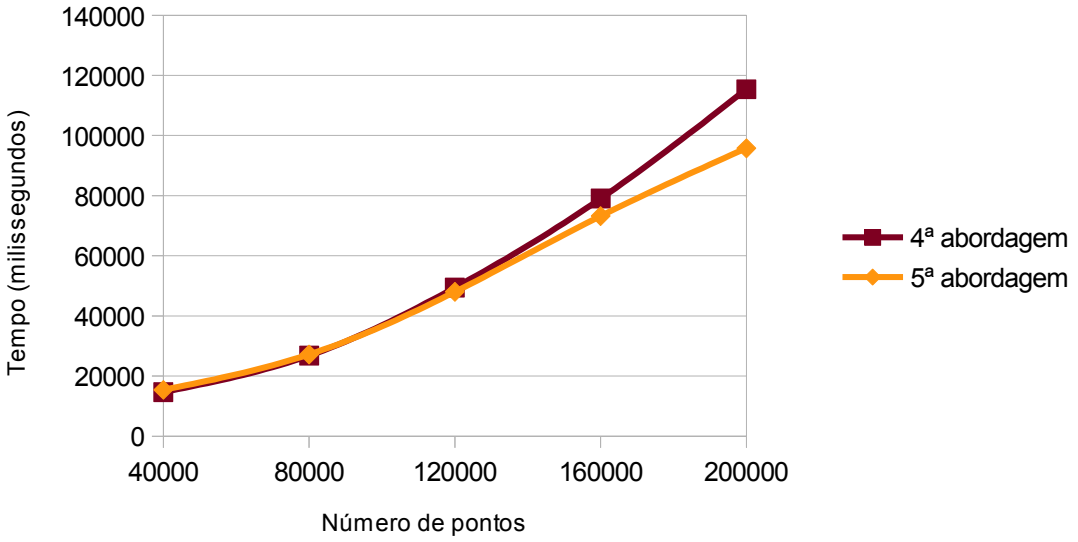


Figura 3.16: Tempo de processamento entre as abordagens 4 e 5 (Wp =50% e Wb=50%).

Embora o tempo de processamento tenha aumentado em ambas as abordagens, devido ao maior número de expansões efetuadas para abranger outras dimensões que não as coordenas espaciais, a curva de crescimento do algoritmo resultante desta abordagem é menos acentuada. Sendo assim, para um peso de 50% para a dimensão espacial na função de similaridade, e para 200.000 pontos, o algoritmo desta abordagem demorou 95 segundos a processar todos os pontos, enquanto o algoritmo na abordagem anterior demorou 115 segundos.

Cada nova abordagem aproveitou as melhores características das abordagens anteriores, sendo a abordagem 5, por isso, a mais matura, e naturalmente a mais eficiente. Como é possível perceber pelos tempos resultantes dos processamentos de teste, a abordagem 5 é significativamente mais eficiente que o SNN original para dados sobre movimento.

A aplicação resultante da implementação do algoritmo SNN descrita no inicio deste capítulo foi sendo alterada à medida que era necessário testar cada nova abordagem, no entanto, a aplicação teve de ser repensada de forma a tornar-se uma ferramenta de clustering com melhor usabilidade para o analista, possibilitando configurações, de forma a minimizar a necessidade de manipular o código fonte da aplicação.

No próximo capítulo descreve-se a nova aplicação do ponto de vista do utilizador. Será, portanto, dada maior ênfase à interface e a outras configurações relacionadas com as funções de distância necessárias ao algoritmo.

Capítulo 4
Aplicação de *Clustering* Baseada no Algoritmo Desenvolvido

À semelhança da aplicação desenvolvida anteriormente para o algoritmo SNN, foi desenvolvida uma nova aplicação para a utilização do algoritmo resultante do capítulo 3. A sua conceção e implementação teve por base a aplicação para o algoritmo SNN, no entanto, além das alterações evidentemente necessárias na parte respeitante ao algoritmo, houveram também algumas preocupações no que concerne a utilização.

Esta aplicação será descrita neste capítulo, tendo em conta, em primeiro lugar, a sua interface, e depois serão retratadas algumas configurações necessárias à sua utilização.

4.1 Interface Gráfica

A interface gráfica da aplicação foi desenvolvida de forma a acomodar as funcionalidades descritas anteriormente. Esta é constituída por três separadores, organizados numa ordem que permita um processo contínuo, da esquerda para a direita. Todas as indicações escritas na interface estão em inglês.

Além dos separadores, existe uma secção, sempre visível, no lado inferior da interface, que contém um botão para sair, gravando o estado de todos os campos e

opções na interface, e uma zona de texto que mostrará a fase de cada processamento e o tempo de cada fase. A divisão de funcionalidades principais por separadores permitirá também, eventualmente, adicionar novas funcionalidades à aplicação mais facilmente no futuro. Assim, essas novas funcionalidades poderão utilizar, separadamente, o cálculo de vizinhos através do algoritmo definido neste trabalho.

No primeiro separador, ilustrado na Figura 4.1, é possível definir o ficheiro ou tabela de base de dados, postgreSQL ou MySQL, onde serão retirados os dados sobre movimento para processamento.

Figura 4.1: Interface gráfica - 1º separador.

No segundo separador, denominado por *"Neighbours"*, trata-se das opções de processamento relativas ao cálculo de vizinhos. Este separador encontra-se ilustrado na Figura 4.2.

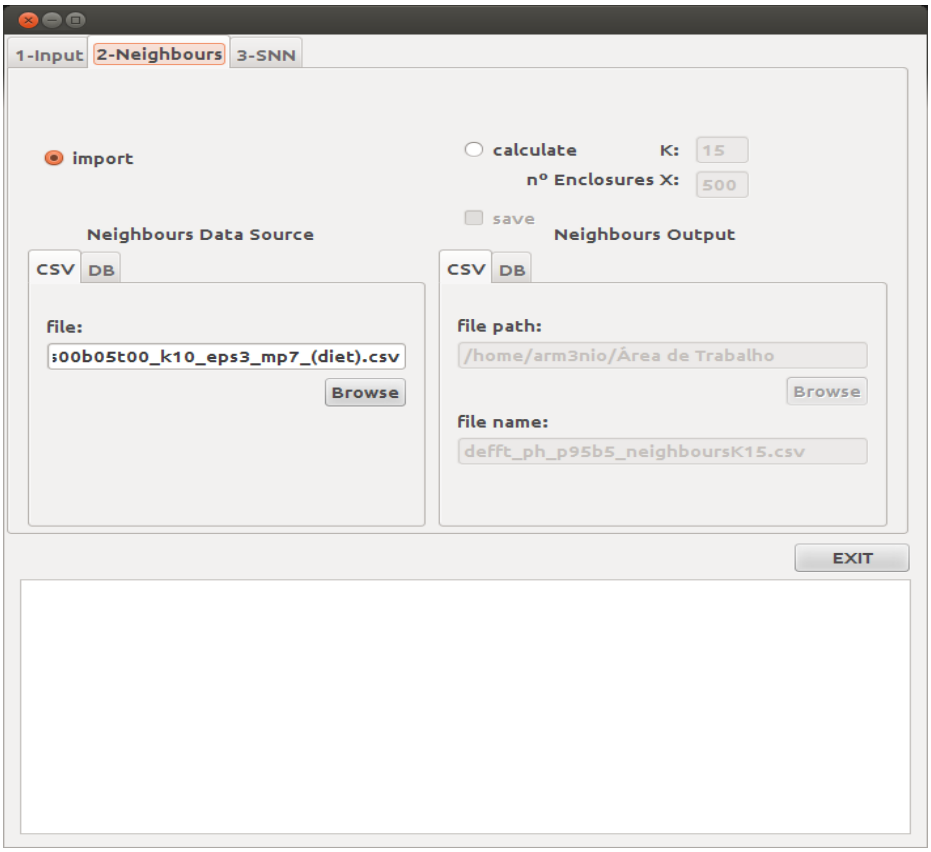


Figura 4.2: Interface gráfica - 2º separador.

É permitida apenas uma entre duas opções. Ou importar a lista de vizinhos mais próximos de cada ponto calculados previamente, ou efetuar esse cálculo quando for iniciado o processo de *clustering* posteriormente, no terceiro separador. Para a segunda opção, será necessário preencher os campos respetivos à variável *K* para o número de vizinhos em cada lista, e ao número de quadrículas que a matriz irá ter no eixo *X* ("nº Enclosures X"). Poder-se-á também escolher guardar a lista de vizinhos, seleccionando-se depois onde os mesmos ficarão.

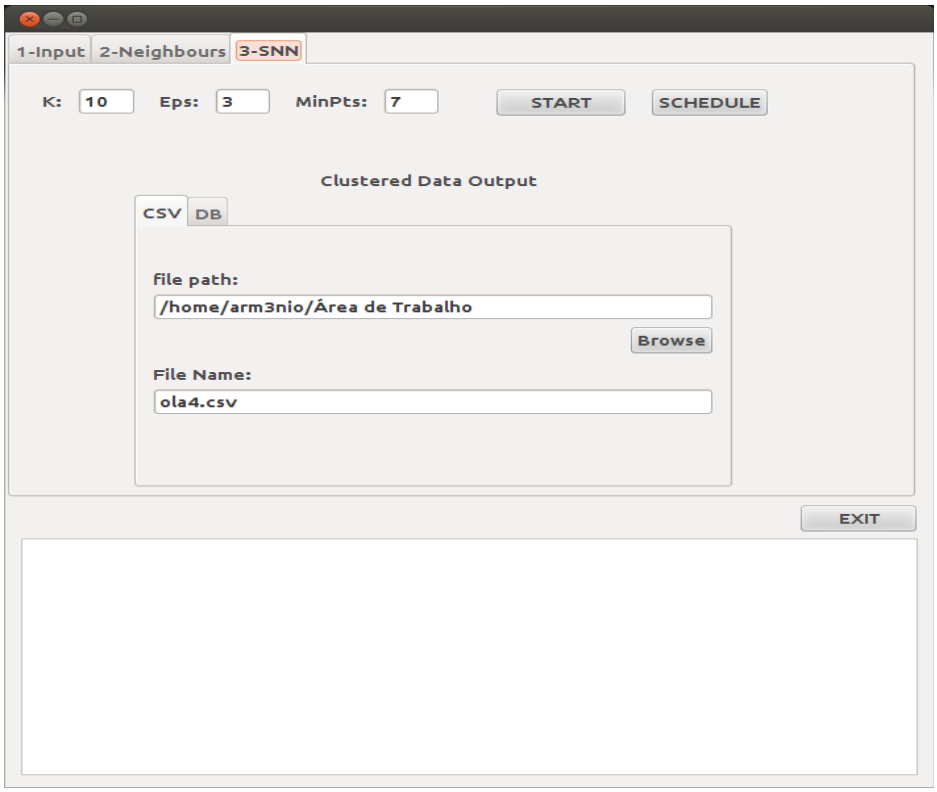


Figura 4.3: Interface gráfica - 3º separador.

O terceiro separador, ilustrado na Figura 4.3, é dedicado ao processo de clustering SNN. Introduzem-se os argumentos próprios deste algoritmo, tratando-se de *MinPts*, *Eps* e *K*, sendo que o valor introduzido de *K* para o *clustering* não poderá ser superior ao valor atribuído a *K* para o cálculo dos vizinhos mais próximos.

No que diz respeito ao destino dos dados após o processamento, pode optar-se entre tabela de base de dados MySQL ou PostgreSQL, ou ficheiro CSV (Comma Separated Values). Para isso escolhe-se o separador respectivo preenchendo-a com os dados necessários.

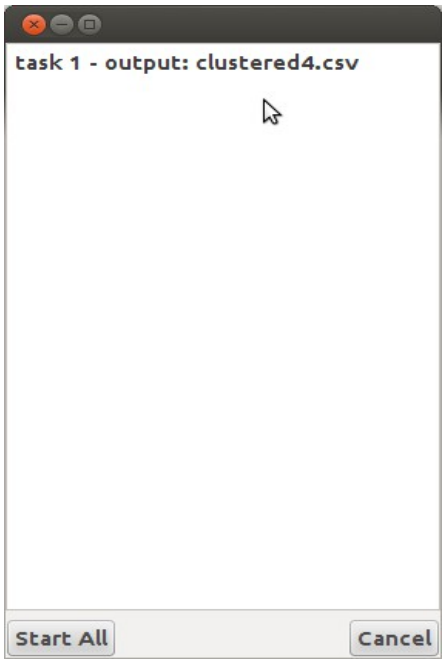


Figura 4.4: Janela de agendamento de processos de clustering.

Quando os dados estão satisfatoriamente introduzidos, poder-se-á clicar no botão "START", dando-se início ao processamento. Todos os dados introduzidos são guardados, e serão carregados automaticamente quando a aplicação for aberta novamente. Alternativamente, pode-se clicar no botão "SCHEDULE" e o processamento será agendado, ficando em fila de espera e visível na janela ilustrada na 4.4. É então possível voltar a editar qualquer um dos campos em todos os separadores e agendar novas tarefas. Clicando no botão "Start All" na janela da Figura 4.4 dá-se início a todos os processamentos agendados.

Do ponto de vista do utilizador, importa salientar que existem configurações que devem ser efetuadas previamente à utilização desta aplicação. Esta tarefa será descrita de seguida.

4.2 Configurações Prévias à Execução da Aplicação

Para manter uma maior flexibilidade de utilização da aplicação, é necessário algum esforço de configuração da mesma previamente. Esta configuração diz respeito à função de similaridade, seja para criar um nova, ou para optar por uma diferente, seja nova ou não. É também possível fornecer variáveis, geralmente relacionadas com o peso atribuído a cada dimensão dos dados. De seguida serão abordadas as formas de configuração definidas para esta aplicação.

Nova função de similaridade

Geralmente, para cada *dataset* ou estudo pretendido, poderá ser necessário utilizar uma nova função de distância. Esta função deve ser definida como uma nova classe, no documento java referente à classe Distance no código fonte da aplicação. Esta nova classe deve implementar o interface DistanceInterface que define cinco métodos:

1. **dist(Point a, Point b);**

   Este método deverá devolver um número do tipo *double*, que, em princípio, deverá corresponder à função que será usada no algoritmo SNN para cálculo da similaridade entre elementos de dados.

2. **distGeo(Point a, Point b);**

   O método distGeo, devolve apenas a distância euclidiana entre dois pontos, considerando apenas a dimensão espacial. Também este método deverá devolver um elemento de tipo *double*.

3. **enclosuresOverflow(ArrayList points, CalcEnclosures cde);**

   Este método servirá especificamente para o cálculo de similaridades em que sejam levadas em conta mais que duas dimensões. A determinada altura do processamento do algoritmo, será necessário calcular a quantos blocos mais se irá estender o cálculo de similaridades, isto para cada bloco no plano bidimensional particionado no algoritmo desenvolvido. Assim será possível não excluir possíveis pontos mais similares, em que

essa similaridade é aumentada por uma ou mais dimensões diferentes à do plano espacial. Assim sendo, este método deverá devolver um número inteiro maior ou igual a zero.

4. **posX();**

   Este método deverá devolver o valor de uma variável da função de distância que guarda a posição da dimensão que irá ser particionada pelo eixo *X*, no vetor de dados de cada ponto. Por exemplo, a abstração de um vetor de movimento numa base de dados pode ser constituída por um índex, longitude, latitude, e *bearing*. Na posição 0 estaria o índex, na posição 1 estaria a longitude, na 2 a latitude, e na posição 3 estaria o bearing. Este método retornaria o número 1, dado que seria a dimensão que quereria seccionar pela matriz no eixo *X*.

5. **posY();**

   Este método funcionará analogamente ao anterior, mas para a dimensão a ser seccionada no eixo *Y*.

Após a definição de uma classe para implementar uma nova função de similaridade, será necessário criar mais uma condição no método de inicialização da classe existente *Distance*. Desta forma torna-se possível definir um novo algarismo que, ao ser introduzido como número de função no ficheiro de configurações apropriado, a classe instanciada será a que foi criada. O código responsável por condicionar a escolha da classe de similaridade encontra-se ilustrado na Figura 4.5.

```
public Distance(ArrayList param) {
    funcNum = (String) param.get(0);

    if (funcNum.equals("1")) {
        distFunc = new Distancia1(param);
    }
    if (funcNum.equals("2")) {
        distFunc = new Distancia2(param);
    }
    if (funcNum.equals("3")) {
        distFunc = new Distancia3(param);
    }
    if (funcNum.equals("4")) {
        distFunc = new Distancia4(param);
    }
    if (funcNum.equals("5")) {
        distFunc = new ClassNovaSimilaridade(param);
    }
}
```

Figura 4.5: Método de inicialização da classe Distance

Selecção da função de similaridade

Quando a função de similaridade já estiver definida na classe *Distance*, poderá ser necessário apenas optar pela mesma no ficheiro de configurações "distance.conf", existente na raiz de ficheiros da aplicação java. Neste ficheiro, além de se optar pela função de similaridade, também é possível introduzir valores a serem utilizados como argumentos na função de similaridade. A quantidade dos argumentos é variável, embora os nomes utilizados para cada argumento, no ficheiro de configurações, tenham de seguir algumas regras descritas de seguida. Na Figura 4.6 é possível ver um exemplo do conteúdo do ficheiro de configurações "distance.conf".

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>selecção da função distancia e seus parametros</comment>
<entry key="parametro_3">0.10</entry>
<entry key="parametro_2">0.0</entry>
<entry key="parametro_1">0.90</entry>
<entry key="FunctionNumber">5</entry>
<entry key="parametro_4">0.0</entry>
</properties>
```

Figura 4.6: Conteúdo do ficheiro "distance.conf"

Tal como é observado no exemplo da Figura 4.6, o argumento de chave "*FunctionNumber*" será o algarismo inteiro que, no método de inicialização da classe *Distance* ilustrado na Figura 4.5, irá determinar qual a classe de similaridade a ser instanciada. Os restantes argumentos terão, obrigatoriamente, de ter como chave "parametro_" seguido de um algarismo continuo aos restantes.

Uma classe de similaridade terá sempre, por imposição do interface que implementam, de receber uma lista na sua inicialização. Esta lista tem, na posição zero, o número da função escolhida. Nas restantes posições terá o parâmetro definido no ficheiro de configuração, cujo número iguale a sua posição nessa lista. Por exemplo, o argumento de chave "parametro_5" do ficheiro "distance.conf" estará na posição 5 da lista fornecida para a instanciação da classe escolhida.

Esta aplicação foi validada experimentalmente, comparando resultados de diferentes análises ao *dataset*, com os resultados da implementação do SNN original.

Capítulo 5
Conclusões

Este capítulo tem o objetivo de concluir este documento. Serão sintetizadas as ilações deste trabalho, e serão reunidas e ponderadas as contribuições e limitações do mesmo. Também serão deixadas algumas propostas e recomendações relativas a possível investigação futura no âmbito deste estudo.

5.1 Síntese do Trabalho Realizado

Neste trabalho, em primeiro lugar, foi enquadrado o problema da falta de eficiência do algoritmo SNN no trabalho de análise de dados sobre movimento. De seguida, foi desenvolvida uma implementação deste algoritmo, que foi testada de forma a identificar os seus processos mais ineficientes, que se provou ser o cálculo de vizinhos. Serviu também para comparar o desempenho das novas abordagens a este algoritmo desenvolvidas, com o desempenho do algoritmo SNN original.

Noutra fase foram desenvolvidas, num processo evolutivo, cinco abordagens ao algoritmo SNN para melhorar o tempo de processamento de dados por parte do mesmo. Estas abordagens foram testadas, sempre no mesmo ambiente, utilizando-se uma função de distância já especificada para a aplicação do SNN original. Estas abordagens basearam-se num conceito de divisão dos pontos por uma matriz, de acordo com as suas

coordenadas espaciais. Os melhores resultados de tempo de processamento foram atingidos quando um maior número de células dessa matriz deixou de significar uma maior probabilidade de erro nos resultados do *clustering*. Isto permitiu aumentar muito a quantidade de células desta matriz, diminuindo assim o seu tamanho, o que se revelou muito importante para a eficiência do algoritmo.

O objetivo deste trabalho era diminuir o tempo de processamento de *clustering* baseado em densidade de pontos, mais especificamente, o tempo de processamento do algoritmo de *clustering* SNN. Este objetivo foi atingido com sucesso, para a função de distância definida.

Finalmente, utilizando o algoritmo desenvolvido, foi implementada e apresentada uma aplicação para permitir a exploração de dados sobre movimento através da nova versão do algoritmo que emergiu da abordagem 5.

5.2 Limitações

O sucesso do processo de *clustering* no algoritmo desenvolvido depende do número de expansões necessárias para abranger todas as dimensões além das coordenadas espaciais, que são também utilizadas na função de distância. Embora já tenha sido implementada uma função para retornar o número de expansões relativas ao *bearing*, qualquer nova função de distância significa um esforço extra para construir uma nova função que retorne o número de expansões relativas às dimensões diferentes das coordenadas espaciais.

É necessário perceber previamente para cada dataset, dependendo da concentração dos seus pontos, qual o valor de *Mx* (número de quadrículas da matriz utilizada no novo algoritmo, no eixo X) mais indicado. No entanto, segundo os testes efetuados para diferentes valores desta variável, não parece que seja um valor muito específico, podendo assumir uma grande amplitude de valores, pelo menos para o *dataset* testado.

O tempo de processamento do novo algoritmo não pode ser previsto através, simplesmente, do número de pontos presentes no *dataset* analisado. Este tempo vai depender da distribuição de pontos, a nível espacial, nesse *dataset*. O algoritmo criado é, ainda, tanto menos eficiente quanto menor for o peso da distância espacial no cálculo da proximidade entre os pontos.

A complexidade do algoritmo não deixa de ser $O(n^2)$ no pior caso, embora para ocaso testado a curva de crescimento do tempo de processamento, relativa ao número de pontos analisados, seja agora muito menos acentuada, tendo em conta a função de distância utilizada.

Falta, ainda, perceber melhor o comportamento deste algoritmo perante diferentes parâmetros de entrada ou pesos das dimensões utilizadas, e perante diferentes conjuntos de dimensões.

5.3 Contribuições

Embora a complexidade do algoritmo criado continue a ser de $O(n^2)$, a curva de crescimento do tempo de processamento, relativa ao número de pontos analisados, é agora muito menos acentuada, tendo em conta a função de distância utilizada. Isto poderá ser traduzido num passo em frente relativamente à análise dos grandes *datasets* de dados sobre movimento, gerados hoje em dia.

Foi atingido o objetivo de diminuir o tempo de processamento do algoritmo SNN, permitindo um trabalho de análise, que geralmente envolve vários processamentos, muito mais célere.

Ainda no sentido de melhorar as condições de análise de dados sobre movimento, foi desenvolvida uma aplicação cujo intuito passa não só por providenciar melhor usabilidade do algoritmo desenvolvido, como permitir que sejam implementadas outras funcionalidades, nesta mesma aplicação, que tirem partido da estratégia desenvolvida para o cálculo de vizinhos.

5.4 Trabalho Futuro

A aplicação final foi desenvolvida separando a primeira fase do algoritmo SNN, o cálculo dos K vizinhos mais próximos de cada ponto, das restantes duas fases deste algoritmo anteriormente descritas. Isto possibilita o desenvolvimento de novas funcionalidades utilizando a estratégia de cálculo de vizinhos aqui desenvolvida. Funcionalidades estas como tratamento de dados ou outros algoritmos baseados em densidades de pontos que poderão ser integrados na aplicação.

A divisão do espaço numa matriz poderá ser uma boa estratégia, também para a gestão de memória, sendo um aspeto que será necessário abordar relativamente à eficiência do SNN e que não foi abordado neste trabalho.

A criação da estrutura de pontos divididos numa matriz através das suas coordenadas espaciais poderá também ser uma boa estratégia para permitir clustering incremental. Isto é, o fato de existir um *dataset* com *clusters* previamente identificados, cujos pontos já se encontram divididos na estrutura da matriz, poderá facilitar a inserção dinâmica nos *clusters* de novos pontos pertencentes ao espaço abrangido pelo *dataset*.

Será vantajoso também, no futuro, automatizar o ajuste dos parâmetros de entrada do algoritmo SNN, não descurando o novo parâmetro *Mx* que definirá o tamanho da matriz onde os pontos serão distribuídos.

Referências

Alvares, L. O., Bogorny, V., de Macedo, J. A. F., Moelans, B., & Spaccapietra, S. (2007). Dynamic modeling of trajectory patterns using data mining and reverse engineering. *Tutorials, posters, panels and industrial contributions at the 26th international conference on Conceptual modeling - Volume 83*, ER '07 (pp. 149–154). Darlinghurst, Australia, Australia: Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1386957.1386981

Andrienko, G., Andrienko, N., Rinzivillo, S., Nanni, M., Pedreschi, D., & Giannotti, F. (2009). Interactive visual clustering of large collections of trajectories. *IEEE Symposium on Visual Analytics Science and Technology, 2009. VAST 2009* (pp. 3 –10). Presented at the IEEE Symposium on Visual Analytics Science and Technology, 2009. VAST 2009. doi:10.1109/VAST.2009.5332584

Andrienko, Gennady, Andrienko, N., & Wrobel, S. (2007). Visual analytics tools for analysis of movement data. *SIGKDD Explor. Newsl.*, *9*(2), 38–46. doi:10.1145/1345448.1345455

Andrienko, N., Andrienko, G., Pelekis, N., & Spaccapietra, S. (2008). Basic concepts of movement data. *Mobility, Data Mining and Privacy-Geographic Knowledge Discovery, Springer, Berlin*, 15–38.

Baraldi, A., & Blonda, P. (1999). A survey of fuzzy clustering algorithms for pattern recognition. I. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *29*(6), 778–785. doi:10.1109/3477.809032

Berkhin, P. (2006). A survey of clustering data mining techniques. *Grouping Multidimensional Data*, (c), 25–71.

Bhavsar, H. B., & Jivani, A. G. (2009). The Shared Nearest Neighbor Algorithm with

Enclosures (SNNAE). *2009 WRI World Congress on Computer Science and Information Engineering* (Vol. 4, pp. 436–442). Presented at the 2009 WRI World Congress on Computer Science and Information Engineering, IEEE. doi:10.1109/CSIE.2009.997

Chen, L., Özsu, M. T., & Oria, V. (2005). Robust and fast similarity search for moving object trajectories. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05 (pp. 491–502). New York, NY, USA: ACM. doi:10.1145/1066157.1066213

Ertoz, L., Steinbach, M., & Kumar, V. (2002). A new shared nearest neighbor clustering algorithm and its applications. *Workshop on Clustering High Dimensional Data and its Applications at 2nd SIAM International Conference on Data Mining* (pp. 105–115).

Ertöz, L., Steinbach, M., & Kumar, V. (2003). Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data (Vol. 47).

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise (pp. 226–231).

Giannotti, F., Nanni, M., Pinelli, F., & Pedreschi, D. (2007). Trajectory pattern mining. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07 (pp. 330–339). New York, NY, USA: ACM. doi:10.1145/1281192.1281230

Guha, S., Rastogi, R., & Shim, K. (1998). CURE: an efficient clustering algorithm for large databases. SIGMOD '98 (pp. 73–84). New York, NY, USA: ACM. doi:10.1145/276304.276312

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM computing surveys (CSUR)*, *31*(3), 264–323.

Karypis, G., Han, E. H., & Kumar, V. (1999). Chameleon: Hierarchical clustering using

dynamic modeling. *Computer*, *32*(8), 68–75.

Kotsiantis, S., & Pintelas, P. (2004). Recent advances in clustering: A brief survey. *WSEAS Transactions on Information Science and Applications*, *1*(1), 73–81.

Lee, J.-G., Han, J., & Whang, K.-Y. (2007). Trajectory clustering: a partition-and-group framework. *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07 (pp. 593–604). New York, NY, USA: ACM. doi:10.1145/1247480.1247546

Li, X., Jiang, S.-Y., & Su, X.-K. (2009). A novel fast clustering algorithm (Vol. 4, pp. 284–288).

Mennis, J., & Guo, D. (2009). Spatial data mining and geographic knowledge discovery–An introduction. *Computers, Environment and Urban Systems*, *33*(6), 403–408.

Meratnia, N., & de By, R. A. (2002). Aggregation and comparison of trajectories. *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, GIS '02 (pp. 49–54). New York, NY, USA: ACM. doi:10.1145/585147.585158

Miller, H. J., & Han, J. (2009). *Geographic data mining and knowledge discovery*. CRC.

Moreira, A., Santos, M. Y., & Carneiro, S. (2005). Density-based clustering algorithms–DBSCAN and SNN. *Available at: get. dsi. uminho. pt/local/download/SNN&DBSCAN. pdf*.

Niu, X.-X., Yang, K.-H., & Fu, D. (2010). Local agglomerative characteristics based clustering algorithm (Vol. 4, pp. 1643–1646).

Rinzivillo, S., Pedreschi, D., Nanni, M., Giannotti, F., Andrienko, N., & Andrienko, G. (2008). Visually Driven Analysis of Movement Data by Progressive Clustering. *Information Visualization*, *7*(3-4), 225–239. doi:10.1057/palgrave.ivs.9500183

Santos, M. Y., Silva, J. P., Moura-Pires, J., & Wachowicz, M. (2012). Automated Traffic Route Identification Through the Shared Nearest Neighbour Algorithm. In J. Gensel, D. Josselin, D. Vandenbroucke, W. Cartwright, G. Gartner, L. Meng, & M. P. Peterson (Eds.), *Bridging the Geographic Information Sciences*, Lecture Notes in Geoinformation and Cartography (pp. 231–248). Springer Berlin Heidelberg. Retrieved from http://www.springerlink.com/content/k35u57483714476h/abstract/

Tripathy, A., Maji, S. K., & Patra, P. K. (2011). FDCA: A fast density based clustering algorithm for spatial database system (pp. 21–26).

Vlachos, M., Kollios, G., & Gunopulos, D. (2002). Discovering similar multidimensional trajectories. *18th International Conference on Data Engineering, 2002. Proceedings* (pp. 673 –684). Presented at the 18th International Conference on Data Engineering, 2002. Proceedings. doi:10.1109/ICDE.2002.994784